ARMY RESEARCH LABORATORY

The Inertial Reticle Technology (IRT) Applied to a .50-cal. M2 Heavy-Barrel Machine Gun Firing From a High-Mobility Multipurpose Wheeled Vehicle (HMMWV)

by Timothy L. Brosseau, Mark D. Kregel,
Bailey T. Haug, and John T. McLaughlin

ARL-TR-2210 April 2000

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5066

ARL-TR-2210 **April 2000**

The Inertial Reticle Technology (IRT) Applied to a .50-cal. M2 Heavy-Barrel Machine Gun Firing From a High-Mobility Multipurpose Wheeled Vehicle (HMMWV)

Timothy L. Brosseau, Mark D. Kregel, Bailey T. Haug,
and John T. McLaughlin
Weapons and Materials Research Directorate, ARL

Abstract

Motion of the muzzle of a weapon fired from a moving vehicle occurs during firing because of many factors, such as vibrations caused by the vehicle's wheels or the terrain. This motion can have adverse effects on the capabilities of the weapon to hit a target, because the shooter is unable to accurately position the muzzle of the weapon onto the target as the projectile exits the barrel. Large, heavy vehicles, such as the Abrams tank, the Bradley Fighting Vehicle, and the costly Apache helicopter, have very expensive gun turrets that are controlled by very expensive, fully stabilized gun sights to accurately position the muzzle of the weapon onto the target. However, small and lightweight vehicles, such as a small helicopter, a fast attack vehicle, or a high-mobility multipurpose wheeled vehicle (HMMWV), cannot justify such expensive gun turrets and fully stabilized sights. Therefore, to improve the accuracy of a weapon firing from a small, lightweight vehicle, the U.S. Army Research Laboratory (ARL) has developed the Inertial Reticle Technology (IRT).

This report presents the complete details of how the IRT was applied to a .50-cal. M2 heavy-barrel machine gun firing from a HMMWV, along with an analysis of stationary and moving vehicle live fire test data. Also presented are analyses of moving vehicle live fire test data from a .50-cal. M2 heavy-barrel machine gun firing from the swivel turret of a standard HMMWV, without the IRT.

Table of Contents

INTENTIONALLY LEFT BLANK.

List of Figures

List of Tables

INTENTIONALLY LEFT BLANK.

1. Introduction

Motion of the muzzle of a weapon fired from a moving vehicle occurs during firing because of many factors, such as vibrations caused by the vehicle's wheels or the terrain. This motion can have adverse effects on the capabilities of the weapon to hit a target because the shooter is unable to accurately position the muzzle of the weapon onto the target as the projectile exits the barrel. Large, heavy vehicles, such as the Abrams tank, the Bradley Fighting Vehicle, and the costly Apache helicopter, have very expensive gun turrets that are controlled by very expensive, fully stabilized gun sights to accurately position the muzzle of the weapon onto the target. However, small and lightweight vehicles, such as a small helicopter, a fast attack vehicle, or a high-mobility multipurpose wheeled vehicle (HMMWV), cannot justify such expensive gun turrets and fully stabilized sights. Therefore, to improve the accuracy of a weapon firing from a small, lightweight vehicle, the U.S. Army Research Laboratory (ARL) has developed the Inertial Reticle Technology (IRT).

2. The IRT Applied to a Weapon Firing From a Lightweight Vehicle

Two test beds were built as part of the IRT program, which was an Army Science and Technology Objective culminating in December 1998. The initial test bed was a fast attack vehicle on which the IRT was integrated with a 5.56-mm M16A2 rifle. The second test bed was a HMMWV on which the IRT was integrated with a .50-cal. M2 heavy-barrel machine gun. This section discusses the common aspects of the design, and the rest of this report focuses on the HMMWV test bed.

The IRT replaces the conventional sight or scope on the weapon fired from a lightweight vehicle with a video camera that is mounted to the weapon and a video display mounted in a remote control panel that rests on the gunner's lap inside the vehicle. The IRT also replaces the weapon mount on the lightweight vehicle with a remotely operated, lightweight, and inexpensive weapon positioner or turret.

In the IRT test beds, the weapon positioner drives the weapon in both the elevation and azimuth directions by means of two inexpensive low-power stepper motors. Two shaft angle encoders are attached to the elevation and azimuth axes of the weapon positioner to measure the relative angular displacements of the weapon relative to the weapon platform. Three orthogonal angular rate sensors are mounted on the weapon platform. The outputs of the rate sensors are integrated to provide the angular displacements of the weapon platform in the pitch, yaw, and roll directions. On the fast attack vehicle, a simple wheel counter, mounted on the vehicle, counts wheel rotations, which allows the calculation of vehicle translation relative to the target. A revolution counter on the speedometer cable performs the same task on the HMMWV.

The initial range of the vehicle to the target is entered into the computer manually or from a range finder mounted on the weapon. A continuous readout of the range is shown on the operator's display in the remote control panel that rests on the gunner's lap inside the vehicle. A small computer is used to generate two electronic pointers that can be overlaid on the video image. The first pointer is a dot that is aligned with the barrel centerline of the weapon and, thus, represents the aim point. With inputs from the range finder, the wheel counter, and the shaft angle encoders, the aim point is ballistically corrected for range and lag angles and becomes the ballistic solution. The second pointer is a crosshair referred to as the inertial reticle. This reticle is driven in opposition to the weapon and vehicle motions, as measured by the shaft angle encoders, integrated rate sensors, and the wheel counter, so that the inertial reticle appears to remain fixed relative to the target, even though the weapon and vehicle might be moving. Even though the video scene may be moving around significantly, there is no relative motion between the inertial reticle and the target. This makes it easy to position the inertial reticle over the desired target using a joystick mounted on the remote control panel.

Once the initial range to the target is put into the computer and the inertial reticle is accurately placed over the desired target, the computer continuously measures the difference in the position of the inertial reticle relative to the aim point. This error signal is used to drive the stepper motors to position the aim point over the inertial reticle. As the aim point and the inertial reticle are aligned, a prediction algorithm on the computer calculates the precise firing time,

ensuring that the projectile exits the muzzle as the ballistic solution is aligned with the inertial reticle. Assuming that the gunner has enabled the system to fire, the precise firing is accomplished by means of an electric solenoid that is attached to the trigger of the weapon. To simplify the display for the gunner, the ballistic solution is typically not displayed and the gunner's tasks are to select the target, enter the range, and keep the inertial reticle on target.

3. The IRT Applied to a .50-cal. M2 Heavy-Barrel Machine Gun Firing From a HMMWV

The IRT was applied to a .50-cal. M2 heavy-barrel machine gun firing from a HMMWV, as shown in Figures 1 and 2. The .50-cal. M2 heavy-barrel machine gun was fitted with a Sony (EVI-330T) CCD camera block. The Sony (EVI-330T) camera block has a 12× optical zoom, auto focus lens. It also has a 2× electronic zoom, which when combined with the 12× optical zoom gives a video image that is equivalent to the image seen through a conventional 12× scope. The Sony (EVI-330T) camera block also has a steady shot (Electronic Image Stabilizer) function that can be easily switched on or off. When the steady shot is switched on, small angular rate sensors in the camera sense camera movement in the elevation and the azimuth directions and electronically reposition the video image to correct for this movement. However, since the inertial reticle must follow the video image, the same amount of video image repositioning being done by the camera to correct for movement must also be applied to the inertial reticle. To determine how much video image repositioning is being done by the camera, a small light source is rigidly placed in front of and off to the side of the camera and projected into the side of the camera lens. The small light source is actually a very small collimating lens attached to one end of a length of fiber optics. A small bulb from a mini-mag light is attached to the other end of the length of fiber optics. The bulb from the mini-mag light produces a small, intense dot of light along the edge of the video image that is very easy to track using a simple tracking algorithm and a frame grabber in the IRT computer. This tracking information was then applied to the inertial reticle to correct for the video image repositioning done by the camera because of camera movement. To ensure that the small dot of light on the video image was always the brightest object in the video scene, several neutral density filters were placed in front of the camera lens

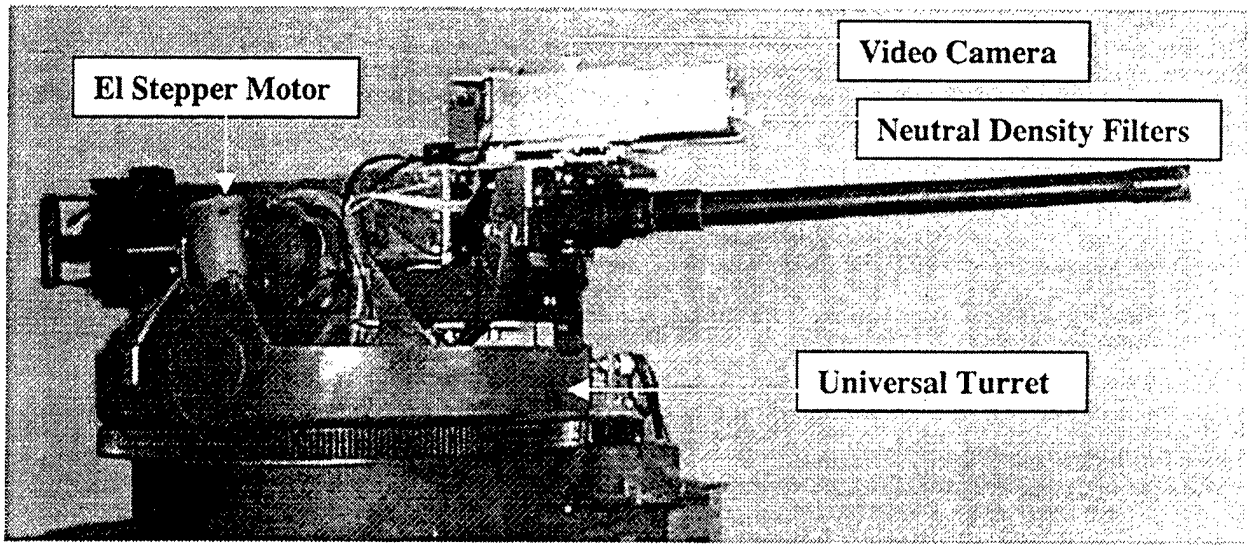

Figure 1. The IRT Applied to a .50-cal M2 Heavy-Barrel Machine Gun Firing From a HMMWV (Right Side View).

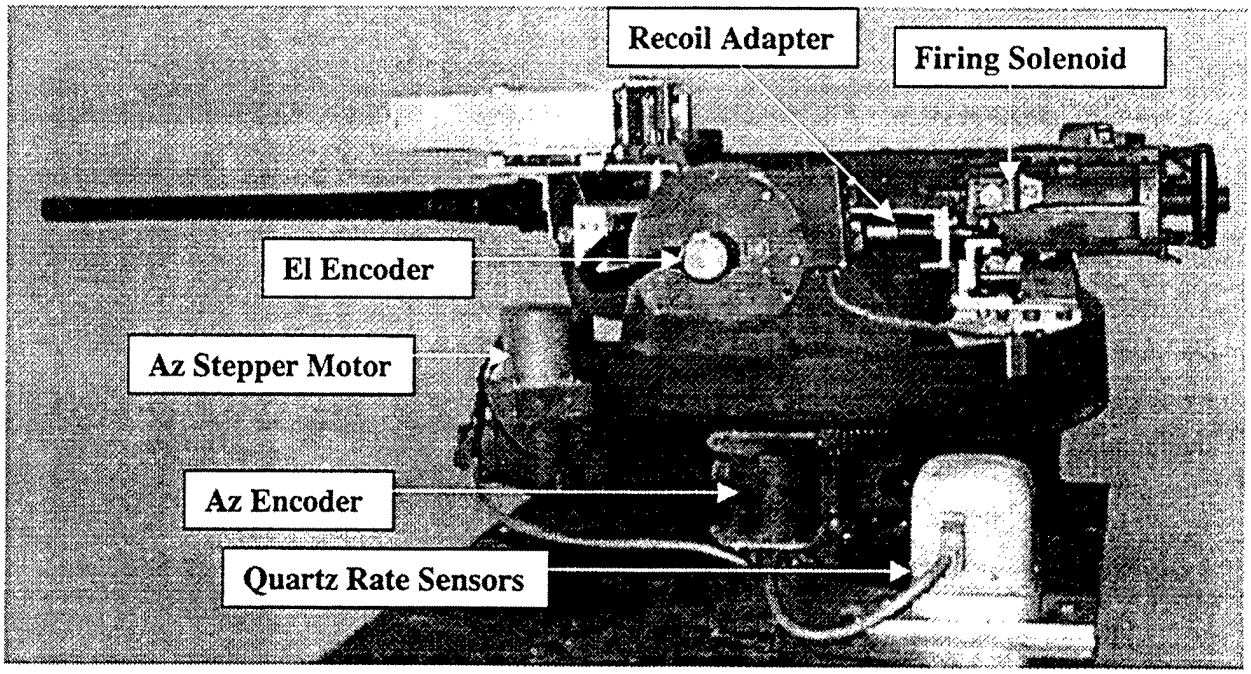

Figure 2. The IRT Applied to a .50-cal M2 Heavy-Barrel Machine Gun Firing From a HMMWV (Left Side View).

and the small collimating lens. This also forced the aperture of the camera lens to remain open even in bright sunlight so that the small dot of light off to the side of the lens was not occluded by the aperture of the camera lens in bright sunlight. The box in front of the camera shown in Figure 1 houses the neutral density filters and keeps stray light out of the camera lens. The video image from the camera block is viewed by the gunner on a Sony flat-panel display monitor (FMD-402A) mounted in the remote control panel.

The .50-cal. M2 heavy-barrel machine gun is mounted in a recoil mount that is attached to a Universal turret from a Cobra helicopter. The recoil mount was designed and built by ARL to reduce the recoil force to the turret, the video camera, the shaft angle encoders, and the quartz rate sensors during firing. The recoil mount allows the receiver of the weapon to recoil about 2 cm during firing of the round and returns the receiver to the full forward position in about 65 ms, which is well before the second round in the burst is fired. The recoil spring adapters for the recoil mount can be seen in Figure 2.

Both the elevation and the azimuth drive motors on the Universal turret were replaced by Superior Electric (M092-FD08) low-power stepper motors that are powered by Industrial Devices (SA Drive) microstepping motor controllers. Two Itek LSI Micro Series (μS/16/23K) shaft angle encoders measure the relative angular displacements between the weapons and the weapon platform in azimuth and elevation. The shaft angle encoders are mounted to the turret, and their output shafts are attached to the elevation and the azimuth axis of the weapon platform. Three Systron Donner quartz rate sensors (QRS-11-0010-101) mounted to the weapon platform measure the angular displacements of the weapon platform in elevation, azimuth, and roll. A quartz revolution counter mounted to the speedomenter cable of the HMMWV measures the vehicle translation.

The initial range of the vehicle to the target is put into the computer manually from inside the vehicle by switches on the remote control panel. Once the initial range to the target is put into the computer and the inertial reticle accurately placed over the desired target, the target designator switch on the gunner's control panel is momentarily engaged. At this time, the initial

position of the target relative to the inertial reticle is determined in inertial coordinates. Theoretically, once the inertial reticle is positioned accurately over the target, it should stay there indefinitely. However, due to random walk in the quartz rate sensors, the inertial reticle will drift off from the target after several seconds. The inertial reticle can be easily repositioned over the target by slight movements of the joystick. If the drift becomes large enough to get outside of the range of the joystick correction, a switch on the gunner's control panel can be turned on, switching the joystick operation from the displacement mode to the velocity mode and giving it an endless correction range. A second switch on the gunner's control panel can also be engaged when the joystick is in velocity mode to cause the stepper motors of the turret to move at a much faster rate, which facilitates getting the target in the field of view.

Control of firing the weapon is accomplished by means of an arm switch and a fire button that are mounted on the gunner's control panel. Once the arm switch is turned on and the gunner is satisfied with the position of the inertial reticle over the desired target, then the gunner depresses the fire button and holds it depressed to enable the electrical firing solenoid. Using the ballistic solution, the weapon automatically fires such that the projectile exit from the barrel occurs when the muzzle of the weapon is properly aligned on the target. The generation of the electronic pointers, the integration of the quartz rate sensors signals, the reading of the wheel counter, the reading of the shaft angle encoders, the determination of the target position in inertial space, the control of the stepper motors, the generation of the predictive fire control algorithm, and the firing of the weapon are all accomplished by a small PC 104 computer and power supply. The complete computer program for the PC 104 computer is given in the Appendix. The gunner's control panel is shown in Figure 3. The video image as seen on the monitor in the gunner's control panel is shown in Figure 4. The range to the target is shown in the upper left corner. The diameter of the black target in the center of the video image is 20.3 cm.

4. Indoor Testing of the IRT Applied to a .50-cal. M2 Heavy-Barrel Machine Gun

Prior to any long-range testing of the IRT applied to a .50-cal. M2 heavy-barrel machine gun firing from a HMMWV, extensive short-range testing was done in the indoor range in building 390 at ARL. Over 100 rounds of M33 ammunition were fired from the weapon with the turret mounted to a rigid plate to determine if the turret, the video camera, the shaft angle encoders, and the quartz rate sensors could withstand the shock from firing. Accuracy measurements were also taken during the initial testing for each round fired. Several rounds of M33 ammunition were also fired with the weapon mounted directly to the rigid plate. The average standard deviations in the elevation and the azimuth directions for several 10-round groups fired from the weapon while it was mounted directly to the rigid plate were .67 mil and .65 mil, respectively. The average standard deviations in the elevation and the azimuth directions for several 10-round groups fired from the weapon with the turret mounted to a rigid plate were .69 mil and .66 mil, respectively. These experiments showed similar results with the weapon mounted in the turret, which was mounted to a fixed plate, and the weapon mounted directly to the plate. It was felt that the IRT integrated with the .50-cal. M2 heavy-barrel machine gun performed substantially as well as a fixed rigid mount, and no more short-range indoor experiments were performed. There was no damage to the turret, the video camera, the shaft angle encoders, or the quartz rate sensors after firing over 100 rounds.

5. Initial Long-Range Outdoor Testing of the IRT Applied to a .50-cal. M2 Heavy-Barrel Machine Gun Firing From a HMMWV

After the indoor testing was completed, the .50-cal M2 IRT test bed was mounted on a HMMWV and initial long-range outdoor firing experiments were done at the U.S. Army Aberdeen Test Center (ATC) H-Field test facility at the Edgewood Area of Aberdeen Proving Ground. The HMMWV selected had been previously used in unrelated experiments and was equipped with a generator, equipment racks, and a platform designed to support this turret. This

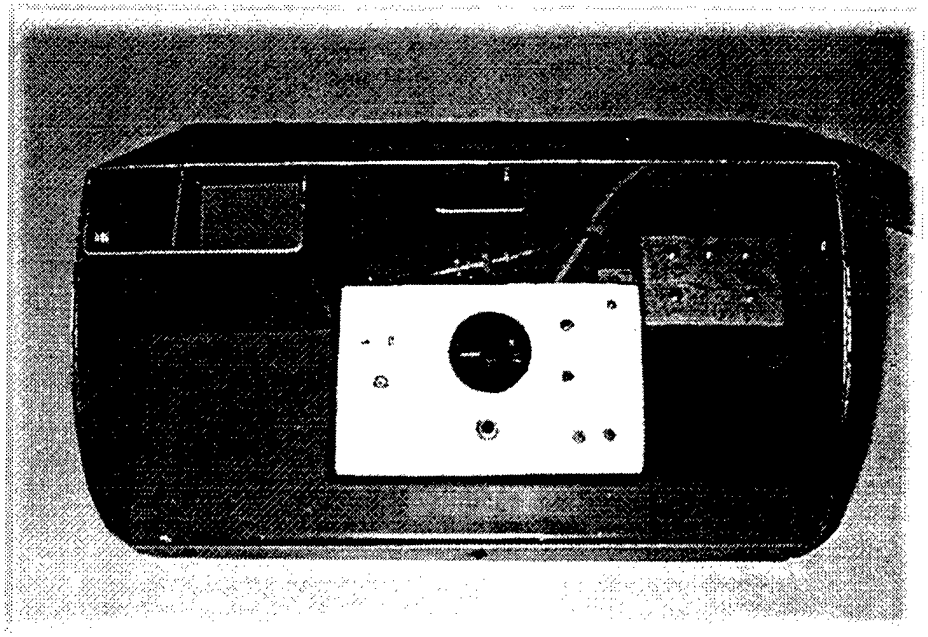

Figure 3. Remote Control Panel (Rests on Gunner's Lap).

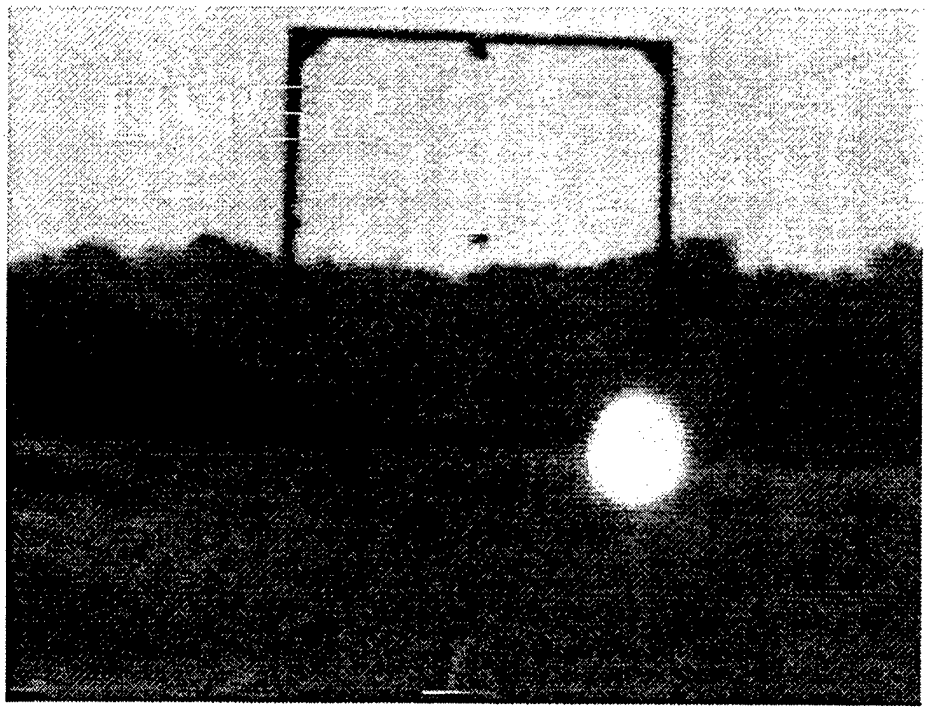

Figure 4. Video Image of a 20.3-cm Target at 452 m (Monitor in Remote Control Panel).

greatly facilitated the ability to experiment with and demonstrate IRT concepts without having to spend limited resources to adapt electronics to a tactical vehicle. There are no technical barriers envisioned in reducing the hardware to a tactical vehicle.

Before any firings from a moving vehicle were done, firings from a stationary vehicle were made at a 400-m target. The average standard deviations in the elevation and the azimuth directions for several 10-round groups of M33 ammunition fired semiautomatically by the gunner from inside the stationary vehicle were .71 mil and .68 mil, respectively. These results, obtained with a gunner inside the stationary vehicle, compared favorably with previous indoor firings. Since mounting the M2 integrated with the IRT on the HMMWV did not result in increased dispersion, it was felt that no further stationary experiments were required, and firing-on-the-move experiments were initiated.

After completing the stationary vehicle firings at the 400-m target, the firings were repeated from a moving vehicle. The initial firing-on-the-move experiments were conducted with the gunner firing from inside the vehicle while the vehicle was moving away from the target. Ten-round groups were fired semiautomatically by the gunner at about 1-s intervals while the vehicle was traveling at 16 kph down a gravel road away from a 400-m target. The average standard deviations in the elevation and the azimuth directions for several 10-round groups of M33 ammunition fired semiautomatically by the gunner from inside the moving vehicle were .74 mil and .70 mil, respectively. Since the average standard deviations were only slightly higher than those for the previous firings from the stationary vehicle, it was felt that the IRT was achieving its maximum accuracy performance for these conditions, and no more semiautomatic firings moving away from the target were made.

Once the firings from the moving vehicle while driving away from the 400-m target were completed, the target was placed 400-m off to the side of the vehicle. The firing experiments were repeated with the gunner firing from inside the moving vehicle and the weapon pointing over the right and left sides of the vehicle while it traversed at 16 kph parallel to the target along a gravel road. The IRT held the inertial reticle on the target and put in the correct amount of lag

angle so that the projectiles hit on target. The average standard deviations for the elevation and the azimuth directions for several 10-round groups of M33 ammunition fired semiautomatically by the gunner from inside the moving vehicle were .75 mil and .73 mil, respectively. The average standard deviations were essentially the same as those for the previous firings for the vehicle traveling straight away from the target.

The next experiment was to determine if the IRT could accurately hit a moving target if the computer was given the target velocity. The first firing tests of firing at a 400-m target while the vehicle was traveling at 16 kph along a gravel road away from the target were repeated. Again the vehicle was driven away from the target at 16 kph from an initial range of 400 m. However, this time during the experiment the target was moving left to right at 16 kph. Ten-round groups were fired semiautomatically by the gunner from inside the vehicle at about 1-s intervals while the vehicle was moving down the gravel road away from the 400-m target. The IRT held the inertial reticle on the target and put in the correct amount of lead angle so that the projectiles hit on target. The average standard deviations in the elevation and the azimuth directions for several 10-round groups of M33 ammunition fired semiautomatically by the gunner from inside the moving vehicle were .77 mil and .79 mil, respectively. These average standard deviations were essentially the same as those for the previous firing for the vehicle traveling straight away from the target.

6. Final Long-Range Outdoor Testing of the IRT Applied to a .50-cal. M2 Heavy-Barrel Machine Gun Firing From a HMMWV

After the initial long-range outdoor testing was completed, a review of the videotape of the inertial reticle taken during the firing tests from the moving vehicle showed that the weapon platform was oscillating at a frequency of about 8 Hz as a result of firing each round of ammunition. The time interval from the fire pulse to projectile exit from the barrel is 28 ms for the hammer-fired .50-cal. M2 heavy-barrel machine gun. This in-bore time is short enough that it was felt the firing predictor should predict well enough at 8 Hz to allow the weapon to be fired

in a burst mode. If the hammer could be re-seared soon enough after firing the first round in the burst, then at 8 Hz, the firing predictor should give a firing pulse every 200 ms, which converts to a rate of fire of 300 rd/min. The sear spring of the weapon was replaced with a much stiffer one and it was determined that with the stiffer sear spring the hammer in the weapon could be easily re-seared in 200 ms. Even though the video scene was blurred for almost 1 s after firing a round, it was felt that the inertial reticle would not drift off from the target a significant amount during the firing of a 3- or 4-round burst.

A final long-range outdoor experiment with the .50-cal. M2 heavy-barrel machine gun IRT test bed firing from a HMMWV was done at the H-Field test facility at the Edgewood Area of Aberdeen Proving Ground. The firing experiments from a moving vehicle done in the initial long-range outdoor testing of the IRT were repeated on the same firing range. However in these experiments, 12 rounds were fired in 3- or 4-round bursts by the gunner from inside the vehicle with about a 1-s interval between each burst while the vehicle was moving at 16 kph down the same gravel road away from the 400-m target. The average standard deviations in the elevation and the azimuth directions for several 12-round groups of M33 ammunition were .83 mil and .80 mil, respectively. In reviewing the videotape of the inertial reticle and the analog tape of the displacements of the weapon taken during the firing tests from the moving vehicle, it was determined that the weapon platform was oscillating at a frequency of about 8 Hz. A check of the firing time data also taken during the firing tests showed that there were no instances when the muzzle of the weapon was not pointing at the target when the projectile exited the gun barrel. Samples of the angular displacements of the weapon taken during the firing of a 3-round burst are shown in Figure 5 along with the firing times for each round in the burst. The average rate of fire of the 3- or 4-round bursts was about 300 rd/min. Since the average standard deviations for the 3- or 4-round burst firings were only slightly higher than the average standard deviations for the semiautomatic firings from the previous final long-range outdoor tests fired from a moving vehicle, it was felt that the IRT applied to a .50-cal. M2 heavy-barrel machine gun firing from a HMMWV was achieving its optimum performance in accuracy for this scenario.

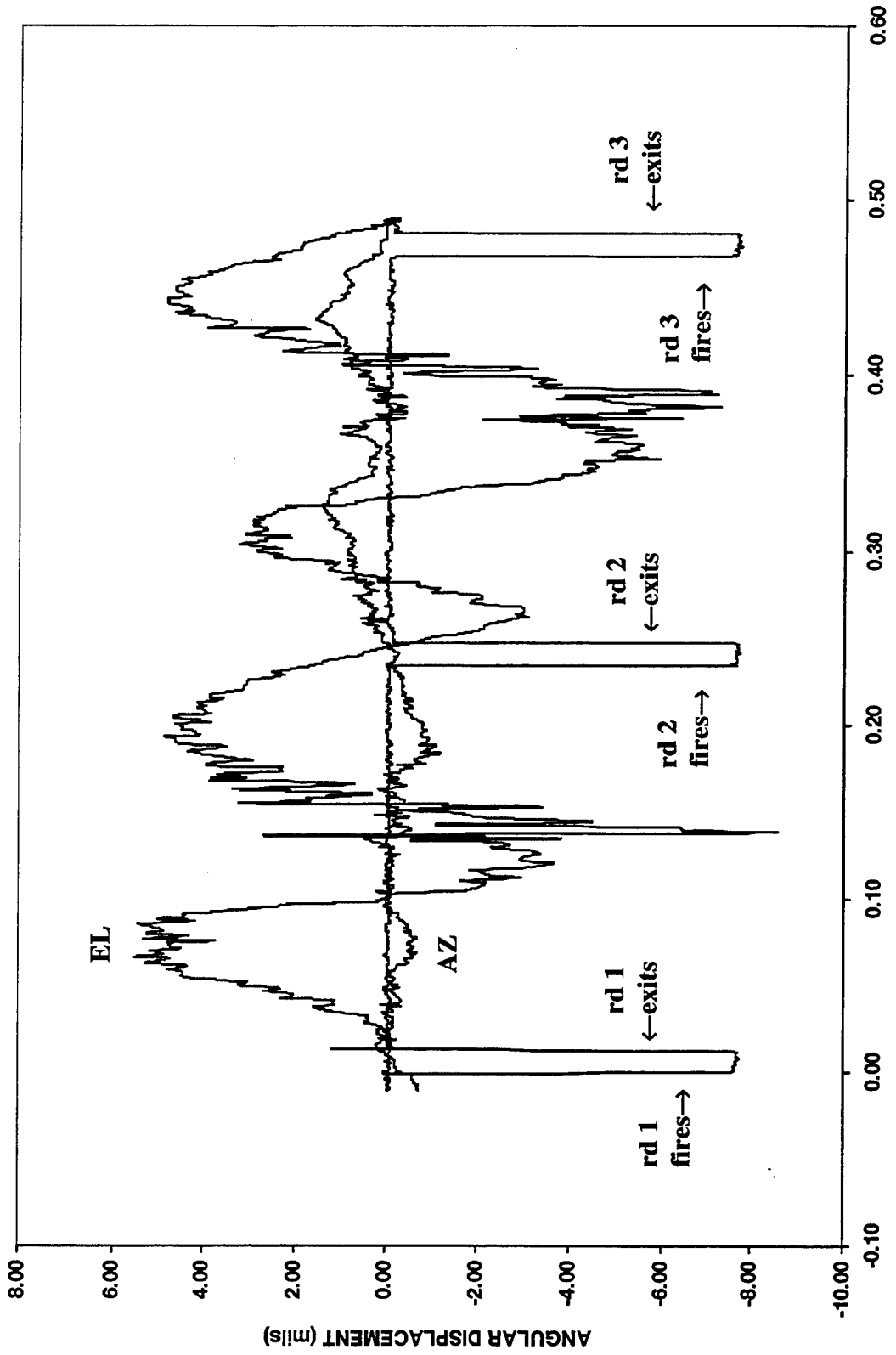

Figure 5. Angular Displacement vs. Time for a Three-Round Burst Fired With the IRT.

7. Long-Range Outdoor Testing of a .50-cal. M2 Heavy-Barrel Machine Gun Firing From a HMMWV

After completing the burst firing experiments, the same experiments were repeated on the same firing range with a .50-cal. M2 heavy-barrel machine gun from a standard swivel mount on a HMMWV. However, in these tests, 20 rounds were fired in 3- or 4-round bursts at about 450 rd/min by a U.S. Army Aberdeen Test Center (ATC) gunner standing behind the weapon with about a 1-s interval between each burst while the vehicle was moving at 16 kph down the same gravel road away from the 400-m target. The average standard deviations in the elevation and the azimuth directions for several 20-round groups of M33 ammunition fired in this experiment were 8.60 mils and 4.50 mils, respectively. In reality, the average standard deviations were actually larger because on the average only 11 of the 20 rounds fired in each group hit the 8-m × 8-m target.

8. Results

The results of the experiments conducted with the .50-cal. M2 heavy-barrel machine gun are summarized in Table 1.

9. Conclusions

As can be seen from Table 1, with the IRT fire control, the engineer was able to shoot .8-m groups, centered on the target, in semiautomatic mode moving away from the target. This was repeated with .9-m groups, centered on target, while the vehicle was traveling a path parallel to the target, again in semiautomatic mode. Although the IRT currently has no means to track the target, by feeding information about target velocity to the computer, the engineer was able to achieve similar results with a crossing target moving at 16 kph.

Table 1. Summary of Results

Firing Scenario	Firing Mode	Standard Deviation		Extreme Spread (m)	Range (m)	Rate of Fire (rd/min)
		Elevation (mil)	Azimuth (mil)			
Stationary vehicle and target	semiautomatic	.71	.68	.7	400	60
Vehicle moving 16 km/hr away from stationary target	semiautomatic	.74	.70	.8	>400	60
Vehicle moving 16 km/hr parallel to stationary target	semiautomatic	.75	.73	.9	400	60
Vehicle moving 16 km/hr away from target moving 16 km/hr left to right	semiautomatic	.77	.79	1.0	>400	60
Vehicle moving 16 km/hr away from stationary target	IRT-controlled burst	.83	.80	1.1	>400	300 (3 rds at 1-s intervals)
Vehicle moving 16 km/hr away from stationary target firing from standard swivel mount	natural rate of fire burst	>8.60	>4.50	>10 (only 11 rds out of 20 fired hit the target)	>400	450 (3 rds at 1-s intervals)

With careful timing, the IRT can control the M2 at a rate of fire of 300 rds/min, approaching the full natural rate of fire and achieve accuracies far in excess of what can be achieved with a conventional weapon under similar conditions. This allows the direction of suppressive fire with deadly accuracy.

The IRT, when integrated with a crew-served weapon, on a lightweight platform, such as a HMMWV or fast attack vehicle, has clearly demonstrated the ability to dramatically improve the accuracy of the weapon system. By demonstrating the ability to fire at high rates of fire, the fire suppressive capability is enhanced with the ability to deliver this type of fire accurately at ranges in excess of 400 m. This technology offers not only the opportunity to achieve accurate fire from a moving platform but to enhance the survivability of the operator by putting the gunner down in the vehicle, reducing his exposure.

INTENTIONALLY LEFT BLANK.

Appendix:

Computer Program*

* Program B0804a is the computer program for the PC 104 computer.

INTENTIONALLY LEFT BLANK.

```
Program B0804a; {June 25, 1998, Version A}
{September 14, 1992, June 21, 1993, March 24, 1994}
{N+}        {March 31, 1994 - Predict 40 ms, cycle 2 ms}
            {April 25, 1994 - used with video reticle genrator}
            {February 7, 1995, February 21, 1995}
            {April 30, 1996, May 8, 1996}
            {May 14, 1996 used with GPS}
  {August 9, 1996 - new processor board (33MhZ), new dt,  new time: 6.0ms}
  {August 9, 1996 - new delays for the Laser Range Finder}
  {October   1996 - Rewritten and updated.  Includes wheel geometry}
  {January   1997 - New predictor - outputted to DAC ports/with firing pulse}
  {November 4,1997 - Camera stabilization}
  {November 20,1997- New camera stabilization}
   {BFCs denotes Body Fixed Coordinates, ICs denotes Inertial Coordinates}

Uses  Crt,Dos;          {Port assignments: 050h to 057h}

Const  fl1=false;fl0=true;    dt = 0.01666/19;
Const  bit0=1;bit1=2;bit2=4;bit3=8;bit4=16;bit5=32;bit6=64;bit7=128;
Const  EarthRadius = 6369537.34;   saeo_c = 0; saep_c= 0;
     WheelToSight_X = 1.2; WheelToSight_Y = 1.0; WheelToSight_Z = 2.0;{BFC}
     ConversionToRadians = 2*pi/65536; NoOfConversionsPerSecond = 3500;
     Coeff1=1.23*(10/9)*(pi/2)/131071; {100 degrees per second > 2**17-1}
     ANG=0.04997558594; wheel_click= 17.0/39.37;
Type
     seal  = double;   bool2 = array[0..7,0..9] of boolean;
     r33   = array[1..3,1..3] of seal; i88 = array[0..7,0..7] of byte;
     r8    = array[0..7] of seal;
     i8    = array[0..7] of byte;
     r3    = array[1..3] of seal;
     ar128b = array[1..150] of byte;
     i3    = array[1..3]   of integer;
     a10   = array[0..10] of seal;
     i42   = array[0..42] of seal;
     a24   = array[0..23] of seal;

VAR      tempp:seal;
      eef,eeg                    :array[0..6] of seal;
      r,v                       :a10;
      video, video_h                  :a24;
      ef, eg                    :i42;
  dont, v0, v0p, h0, v1, h1, v2, h2,vp,hp,ii   :integer;
     fe                        :bool2;
   dig                         : array[0..3] of word;
 xph,yph,xpv,ypv, hr, min, sec, hun, wi, wi_    :word;
```

```
  P6, ww,swz,tee, spot                    :byte;
              PortC0,PortC1,flop,R6,r6p  : byte;
   s                                : ar128b; {Supports the GPS}
  c_azi,c_ele, q,  CosO,SinO,CosP,SInP      : Seal;
kk,  xc,yc, gd, gm, i,m ,e  ,TB, efp        : integer;
     Fir,FirP,Error, GPS, WC, PrintIt              : boolean;
                          nnn       : i8;
too,tpp,tau,  we0,we1,we2,we3,we4,we5,we6      :seal;
  Range, Azi0, Azi1, Ele0, Ele1, speed_       :seal;
   Temp,Slew                        : seal;
  B1,B2,  sum1,  sum2, wx, wy,  wz        : seal;{Body fixed angular rates}
  C_Joy_O, C_Joy_P, P1, P2,temp2,  SE     : seal;
  Port7Bit6, Port7In, BI0,BI1,Bib, sam     : byte;
 cntr,lp, channel, n3, n2, n1, n0, btns    : byte;  {Raw sensor inputs}
 port6, Port9_In, n7,  n6,  n5,  n4      : byte;  {Raw sensor inputs}
 PortB_In,P7InP, j, k, cnt, P7In, PCIn     : byte;  {couters and flag}
 Looper,          Delay,  wrd        : Word;
                       nn     : i88;    {Raw sensor inputs}
          we7,    LLL,  Count       : word;
 c_a_drift, c_e_drift                  : seal;
            predict30             : seal;
 Dist,  Temp_word,app,bpp                 : word;
 SaeO, SaeP,SaeO_, SaeP_,s0,s1,s2,s3,s4   : seal;
 IntO,IntP, JoyO, JoyP, Speed,err,zse      : seal;
vxx, vyy, vzz,dxx,dyy,dzz,dx,dy,dz         : seal;
                    a,aa        : r33;  {Transformation matrix}
            xi,  et,  ze,  ch       : seal;  {Quaterian vriables}
              xidp,etdp,zedp,chdp    : seal;  {Predicted derivatives}
 D_Az0,D_El0,   xid0, etd0, zed0, chd0      : seal;  {Previous derivatives}
 rr, ss,  tt,  xx, yy, zz,       dt2      : seal;  {Normalized time step}
 rrr,sss,ttt,xxx,yyy,zzz,D_Az1,D_El1        : seal;  {Body fixed angular rates}
 xa,ya,za,xb,yb,zb                      : seal;  {Body fixed linear accelerations}

 TimeOfFlight,xw, yw, zw, t              : seal;  {Defines point 1, previous}
 DriftX, DriftY, DriftZ, sum, d_temp       : seal;
 JoyP_,JoyO_,o,oo,ooo,p,pp,ppp,ao,ap       : seal;
 YO9,YP9,YO,YP,SO,SP,john,jane             : seal;
 col ,aff,agg                        : r3;
 the,phi,alt, zzzz                     : real;
 indx                              : i3;
  vc0, hc0,  h0p                       : word;
 V_Initial, H_Initial, vvvv,hhhh                    :word;
 hcp, vcp, vc, hc, vv,hh,  vvc,hhc, v_d,h_d           :integer;
 diff0,diff1,diff2,diff3,diffh3,diffh2,diffh1,diffh0    :integer;
```

```
Procedure PutP3( var A:Byte); begin asm  mov dx,233h; les bx, A; mov al,es:[bx];  out dx,al
end end;
Procedure PutP6( var A:Byte); begin asm  mov dx,236h; les bx, A; mov al,es:[bx];  out dx,al
end end;

Procedure Put6(B:byte;Var A:byte);
 begin a:=b;asm  mov dx,236h; les bx, A; mov al,es:[bx];  out dx,al  end end;

Procedure Put2(B:byte;Var A:byte);
 begin a:=b;asm  mov dx,232h; les bx, A; mov al,es:[bx];  out dx,al  end end;

Procedure Put4(B:byte;Var A:byte);
 begin a:=b;asm  mov dx,234h; les bx, A; mov al,es:[bx];  out dx,al  end end;

Procedure Put3(B:byte;Var A:byte);
 begin a:=b;asm  mov dx,233h; les bx, A; mov al,es:[bx];  out dx,al  end end;


Procedure D;var i:word;
begin
   for i:=0 to 1 do  begin  end;
end;


 Procedure ConfigA(var z:byte);{J3}
  begin asm
  mov dx,0303h;{Configuration A}les bx,z;mov al,es:[bx];out dx,al end end;

  Procedure ConfigB(var z:byte);{J4}
  begin asm
  mov dx,0307h;{Configuration B}les bx,z;mov al,es:[bx];out dx,al end end;

  Procedure ConfigC(var z:byte);{J3}
  begin asm
  mov dx,0333h;{Configuration C}les bx,z;mov al,es:[bx];out dx,al end end;

  Procedure ConfigD(var z:byte);{J4}
  begin asm
  mov dx,0337h;{Configuration D}les bx,z;mov al,es:[bx];out dx,al end end;


Procedure Set_Pix(var V,H,M:word;var N:word);{M offset, N number}
begin asm      {N:0 0 0 0 c c c c  n n n n  n n n n}
 {M=1 next pixel next column over, M=512 next pixel next row down}
 les bx,H
```

```
  mov cx,es:[bx] {cx: 0 0 0 0  0 0 0 b8  b7 b6 b5 b4  b3 b2 b1 b0, H}
  les bx,V
  mov dx,es:[bx] {dx: 0 0 0 0  0 0 0 a8  a7 a6 a5 a4  a3 a2 a1 a0, V}
  rol dx,1       {dx: 0 0 0 0  0 0 a8 a7  a6 a5 a4 a3  a2 a1 a0 0}
  xor al,al      {ax: x x x x  x x x x  0 0 0 0  0 0 0 0}

  mov ah,dl      {ax:a6 a5 a4 a3  a2 a1 a0 0  0 0 0 0  0 0 0 0}
  add cx,ax      {cx:a6 a5 a4 a3  a2 a1 a0 b8  b7 b6 b5 b4  b3 b2 b1 b0}
  mov al,dh      {ax:a6 a5 a4 a3  a2 a1 a0 0  0 0 0 0  0 0 a8 a7}
  and al,03h     {ax:a6 a5 a4 a3  a2 a1 a0 0  0 0 0 0  0 0 a8 a7}
  push ax

  mov dx,236h
  out dx,al  {Output to P0, program the two "page" bits,  P0.1, P0.0}

  les bx,M; mov dx,es:[bx]; {[M] -> dx,    cx dx      reserved}
  les bx,N; mov bx,es:[bx]; {[N] -> bx,    cx dx bx   reserved}
            {bx: 0 0 0 0  c c c c  n n n n  n n n n}
  mov ax,0a000h  {ax: 1 1 0 1  0 0 0 0  0 0 0 0  0 0 0 0}
  mov es,ax      {es: 1 1 0 1  0 0 0 0  0 0 0 0  0 0 0 0}
            {dx: offset, cx dx bl es reserved}

  mov ax,bx      {ax: color and number, cx dx al es reserved}
  mov bx,cx      {bx:a6 a5 a4 a3  a2 a1 a0 b8  b7 b6 b5 b4  b3 b2 b1 b0}
  mov cx,ax      {cx:              color              number}

  pop ax         {ax:a6 a5 a4 a3  a2 a1 a0 0  0 0 0 0  0 0 a8 a7}

@1:  mov es:[bx],ch     {Move print code to video memory}
     add bx,dx;
     jb @4;    {Jump if CY=1 or Z=1}

@5:  dec cl
     jnz @1
     jmp @6

@4:
     inc al
     push dx
     mov dx,0236h
     out dx,al
     pop dx
     mov ah,1;   @7:  dec ah; jnz @7;     {Delay a while}
     jmp @5
@6:
```

```
    end
end;



Procedure LineV( x0, y0,n:word; color:boolean);
var m :word;
begin
        M:=512;
        if not color then N:=N+$0f00;
        Set_Pix(x0,y0,M,N);
end;

Procedure LineH( x0, y0,n:word; color:boolean);
var m :word;
begin       M:=1;
        if not color then N:=N+$0f00;
        Set_Pix(x0,y0,M,N);
   end;


Procedure Put_Digit(var r:seal; var cnt:byte);
var y:word; i:byte;a,b,c:word;
const dx=36;  dy=25;  dx2=dx div 2; x=60; {x component - down \/}
begin
c:=1000;  b:=trunc(abs(r));
for i := 0 to 3 do
begin
   a:= b div c;
   if ( (i=(cnt and 03)) {and (dig[i]<>a)} ) then
   begin
      y:= i*40+60 ; {Y component - over ->}
      LineH(x    ,  y   ,     dy ,   fe[0,a]);
      LineH(x+dx2 ,  y   ,     dy ,   fe[1,a]);
      LineH(x+dx  ,  y   ,     dy ,   fe[2,a]);

      LineV(x    ,  y   ,     dx2,   fe[3,a]);
      LineV(x    ,  y+dy ,    dx2,   fe[4,a]);
      LineV(x+dx2 ,  y   ,     dx2,   fe[5,a]);
      LineV(x+dx2 ,  y+dy ,    dx2,   fe[6,a]);
      dig[i]:=a;
  end;
  b:= b - a*c; c:= c div 10;
 end;
inc(cnt);
```

```
end;

Procedure G_Init;
var i,h,m,n:word;
begin
ww:=$80;ConfigA(ww);
ww:=$82;ConfigB(ww);
ww:=$9b;ConfigC(ww);
ww:=$99;ConfigD(ww);
fe[0,0]:=fl1;fe[1,0]:=fl0;fe[2,0]:=fl1;fe[3,0]:=fl1;fe[4,0]:=fl1;fe[5,0]:=fl1;fe[6,0]:=fl1;
fe[0,1]:=fl0;fe[1,1]:=fl0;fe[2,1]:=fl0;fe[3,1]:=fl0;fe[4,1]:=fl1;fe[5,1]:=fl0;fe[6,1]:=fl1;
fe[0,2]:=fl1;fe[1,2]:=fl1;fe[2,2]:=fl1;fe[3,2]:=fl0;fe[4,2]:=fl1;fe[5,2]:=fl1;fe[6,2]:=fl0;
fe[0,3]:=fl1;fe[1,3]:=fl1;fe[2,3]:=fl1;fe[3,3]:=fl0;fe[4,3]:=fl1;fe[5,3]:=fl0;fe[6,3]:=fl1;
fe[0,4]:=fl0;fe[1,4]:=fl1;fe[2,4]:=fl0;fe[3,4]:=fl1;fe[4,4]:=fl1;fe[5,4]:=fl0;fe[6,4]:=fl1;
fe[0,5]:=fl1;fe[1,5]:=fl1;fe[2,5]:=fl1;fe[3,5]:=fl1;fe[4,5]:=fl0;fe[5,5]:=fl0;fe[6,5]:=fl1;
fe[0,6]:=fl0;fe[1,6]:=fl1;fe[2,6]:=fl1;fe[3,6]:=fl1;fe[4,6]:=fl0;fe[5,6]:=fl1;fe[6,6]:=fl1;
fe[0,7]:=fl1;fe[1,7]:=fl0;fe[2,7]:=fl0;fe[3,7]:=fl0;fe[4,7]:=fl1;fe[5,7]:=fl0;fe[6,7]:=fl1;
fe[0,8]:=fl1;fe[1,8]:=fl1;fe[2,8]:=fl1;fe[3,8]:=fl1;fe[4,8]:=fl1;fe[5,8]:=fl1;fe[6,8]:=fl1;
fe[0,9]:=fl1;fe[1,9]:=fl1;fe[2,9]:=fl1;fe[3,9]:=fl1;fe[4,9]:=fl1;fe[5,9]:=fl0;fe[6,9]:=fl1;
end;

Procedure G_Set( x,y,xp,yp:word);
Const dx=20; dy=30;   dx2=dx div 2; dy2=dy div 2;
Var ymdy2,xmdx2:word;
begin
 xmdx2:=x-dx2; ymdy2:=y-dy2;
 LineV(xpv, ypv, dx, true);
 LineV(xmdx2, y, dx, false);
 xpv:=xmdx2; ypv:=y;

 LineH(xph, yph, dy, true);
 LineH(x, ymdy2, dy, false);
 xph:=x; yph:=ymdy2
end;


 PROCEDURE LuDcmp( n :integer; VAR   a: r33;  VAR indx :i3; VAR  d :seal);
 CONST tiny=1.0e-20;

 VAR k,j,imax,i:integer;
 sum,dum,big:seal;
      vv:r3;
 BEGIN
 ;
 d:=1.0 ;
```

```
 FOR i:=1 TO n DO BEGIN
  big:=0.0;
   FOR j:= 1 TO n DO   IF (abs(a[i,j]) > big) THEN   big:=abs(a[i,j]);
   IF (big=0.0) THEN BEGIN      END; {if}
   vv[i]:=1.0/big;
 END; {for i}
;
 FOR j:= 1 TO n DO BEGIN
  IF (j>1) THEN BEGIN
   FOR i:=1 TO j-1 DO BEGIN
    sum:=a[i,j];
    IF (i>1) THEN BEGIN
      FOR k:= 1 TO i-1 DO sum :=sum-a[i,k]*a[k,j];  a[i,j] := sum;
    END {if i}
   END {for i}
  END; {if j}
;

  big:=0.0;
  FOR i:=j TO n DO BEGIN
     sum:= a[i,j];
     IF (j>1) THEN BEGIN
       FOR k:=1 TO j-1 DO sum:=sum-a[i,k]*a[k,j]; a[i,j]:= sum;
     END; {if j}
   dum:= vv[i]*abs(sum);
   IF (dum>big) THEN BEGIN  big:=dum; imax :=i END {if dum}
  END; {for i}
;
  IF (j<> imax) THEN
     BEGIN
       FOR k:= 1 TO n DO BEGIN
           dum:=a[imax,k]; a[imax,k]:=a[j,k]; a[j,k]:=dum;
       END; {for k}
      d:=-d; vv[imax]:=vv[j];
      END;  {if j}
;
   indx[j]:= imax;
   IF(j<>n) THEN
     BEGIN
        IF(a[j,j]=0.0) THEN a[j,j]:=tiny;  dum :=1.0/a[j,j];
        FOR i := j+1 TO n DO a[i,j]:=a[i,j]*dum;
     END {if j}
    END; {for j}
;
   IF(a[n,n] =0.0) THEN a[n,n] := tiny;
```

```
 END; {proc}


Procedure LuBkSb(n :integer; VAR indx :i3; VAR b :r3; VAR a :r33);
VAR j,ip,ii,i:integer;
   sum:seal;
BEGIN
   ii:=0;
   FOR i:=1 TO n DO BEGIN
      ip:=indx[i]; sum:=b[ip]; b[ip]:=b[i];
      IF (ii <>  0) THEN
         BEGIN
            FOR  j:= ii TO i-1 DO sum:=sum-a[i,j]*b[j];
         END {if ii}
      ELSE     IF (sum <> 0.0) THEN         II:= i;
      b[i]:=sum;
   END; {for i}
   FOR i:= n DOWNTO 1 DO BEGIN
      sum := b[i];
      IF (i<n) THEN FOR j:=i+1 TO n DO sum := sum -a[i,j]*b[j];
      b[i]:= sum/a[i,i];
   END {for i}
END; {procedure}


   Procedure Mat_Inv(var a,y:r33); {Generates the inverse matrix of A in Y}
   var i,j,n:integer;   d:seal;    {The matrix A is destroyed}
   begin
   n:=3;
   LuDcmp(n,a,indx,d);
   for j := 1 to n do
    begin
      for i := 1 to n do  col[i]:=0 ;
      col[j]:=1.0;
      LuBkSb(n,indx,col,a);
      for i:= 1 to n do  y[i,j]:=col[i];
    end;
   end;



  Procedure MATMAT ( var c:r33;  a, b:r33);
  var i ,j,k:word; sum:seal;
  begin
   for i :=1  to 3 do   for k := 1 to 3 do
```

```
 begin sum:=0;for j:=1 to 3 do  sum:=sum+a[i,j]*b[j,k]; c[i,k] := sum end;
end;


Procedure MATMUL(var a,b:r3;var c:r33);
var i,j:word; sum:seal;
begin for i:= 1 to 3 do begin sum := 0;
  for j := 1 to 3 do sum := sum + b[j]*c[i,j];
  a[i] := sum end end;



Procedure MATMULInv(var a,b:r3;var c:r33);
var i,j:word; sum:seal;
begin for i:= 1 to 3 do begin sum := 0;
  for j := 1 to 3 do sum := sum + b[j]*c[j,i];
  a[i] := sum end end;



Procedure VDIF(var a,b,c:r3);
var i:word;begin for i:=1 to 3 do a[i]:=b[i]-c[i] end;


Procedure CMULT( var a,b,c:r3);
begin a[1]:=b[2]*c[3]-b[3]*c[2]; a[2]:=b[3]*c[1]-b[1]*c[3];
    a[3]:=b[1]*c[2]-b[2]*c[1]  end;


Procedure Norm(var a:r3);
var temp:seal; i :word;
  begin temp := 0;
  for i := 1 to 3 do temp := temp + sqr(a[i]); temp := sqrt(temp);
  for i := 1 to 3 do a[i]:=a[i]/temp end;



Procedure DotProd(var r:seal;a,b:r3);
begin r := a[1]*b[1]+a[2]*b[2]+a[3]*b[3]; end;


Function ATan (y,x:real):real;
var  u:real;
begin
 if ( (x=0) and (y=0) ) then atan:=0.0 else
```

```
  begin
   if (abs(x) < abs(y)) then
   begin {abs(x) <  abs(y)}
    u:=arctan(abs(x/y));
    if x<0 then
     begin {x<0} if y>0 then atan:=pi/2+u else atan:=-pi/2-u end else
     begin {x>0} if y>0 then atan:=pi/2-u else atan:=-pi/2+u end
   end else
   begin {abs(x) >= abs(y)}
    u:= arctan(abs(y/x));
    if x<0 then
     begin {x<0} if y>0 then atan:=pi  -u else atan:= -pi +u end else
     begin {x>0} if y>0 then atan :=    u else atan:=     -u end
   end
  end
 end;


Procedure Mat(var xi,et,ze,ch:seal;  var a:r33);
var  ze2, et2, xi2, ch2, ze_et, ze_xi, ze_ch, xi_et, et_ch, xi_ch:seal;
begin  {calculates elements of the transformation matrix}
 ze2 := ze*ze; xi2 := xi*xi; et2 := et*et; ch2 := ch*ch;  et_ch:=et*ch;
 ze_et:=ze*et; ze_xi:=ze*xi; ze_ch:=ze*ch; xi_et:=et*xi;  xi_ch:=xi*ch;

 a[1,1]:=xi2-et2-ze2+ch2;a[1,2]:= 2*(xi_et+ze_ch);a[1,3]:= 2*(ze_xi-et_ch);
 a[2,1]:=2*(xi_et-ze_ch);a[2,2]:=-xi2+et2-ze2+ch2;a[2,3]:= 2*(ze_et+xi_ch);
 a[3,1]:=2*(ze_xi+et_ch);a[3,2]:= 2*(ze_et-xi_ch);a[3,3]:=-xi2-et2+ze2+ch2
end; {procedure}


Procedure Mat_Der(var xi, et, ze, ch, w1, w2, w3, xi_, et_, ze_, ch_:seal);
begin xi_:=( ch*w1 - ze*w2 + et*w3)/2; et_:=( ze*w1 + ch*w2 - xi*w3)/2;
    ze_:=(-et*w1 + xi*w2 + ch*w3)/2; ch_:=(-xi*w1 - et*w2 - ze*w3)/2 end;


Procedure ICsToBFCs(var x, y, z, x_, y_, z_ :seal);
begin  x:=x_*a[1,1]+y_*a[1,2]+z_*a[1,3];
     y:=x_*a[2,1]+y_*a[2,2]+z_*a[2,3];
     z:=x_*a[3,1]+y_*a[3,2]+z_*a[3,3];  end;


  Procedure BFCsToICs (var x_,y_,z_:seal; x,y,z:seal);
  begin  x_:=x * a[1,1] + y * a[2,1] + z * a[3,1];
       y_:=x * a[1,2] + y * a[2,2] + z * a[3,2];
```

```
    z_:=x * a[1,3] + y * a[2,3] + z * a[3,3] end;


  Procedure Initialize_Q (var xi, et, ze, ch :seal);
  begin xi:=0; et:=0; ze:=0; ch:=1; end;



 Procedure Integrate(var a,b,c,d, u0,u1, v0,v1, w0,w1, x0,x1:seal);
 begin a:=a+(u0+u1)*dt2;b:=b+(v0+v1)*dt2;c:=c+(w0+w1)*dt2;d:=d+(x0+x1)*dt2
 end;


 Procedure Up_Date(var a,b,c,d,e,f,g,h:seal);begin a:=b;c:=d;e:=f;g:=h end;


 Procedure Normalize(var a,b,c,d:seal);
 var sum:seal;  begin  {Normalize the Quaterion coefficients}
          sum:= sqrt( sqr(a) + sqr(b) + sqr(c) + sqr(d) );
          a:=a/sum; b:=b/sum; c:=c/sum; d:=d/sum end;



Procedure Step(var value:word);
begin
 asm      {sccc cddd dddd dddd}
  mov dx,0300h;
  les bx,value;   mov ax,es:[bx];   {ah:dddd dddd, al:dddd dddd}
  out dx,al;     {dddd dddd}        {Bits 0 - 7 > 300h}
  mov dx,0302h;          {302h > dx}
  mov al,ah
  out dx,al
 end;
end;


 Procedure Stepper_Driver( Channel:byte; var WP,D:seal; W,S:seal);
 const kappa = 100; {Defines the maximum angular acceleration} var zz,z:seal;
 var sign:boolean; wrd:word; a,b,dw:seal;
 begin
 if channel = 1 then z:=2.4  else z:= 1.2; {channel 1 is azimuth}
  if ((P7In and 1)=0) then w:=0;
  w:=w*z*8.0e5; {One degree error translates to maximum slew rate (w:=1)}

  dw:=w-wp;
```

```
  if dw >  kappa then w:=wp+kappa; if dw < -kappa then w:=wp-kappa;
  if w > 16383 then w:= 16383 else if w <-16383 then w:=-16383;
  wp:=w;
  if w<0 then sign:=false else sign:=true;
  w:=abs(w);
  zz:=3.0;
  if (w <  1) then a:= 16383 else a:= 16383/w; if a<zz then a:=zz;
  b:=trunc(a+d+0.5); d:=d+a-b; wrd := trunc(b);

  if channel = 0    then wrd := wrd + 32768;
  if sign    = true then wrd := wrd + 16384;
  Step (Wrd);
end;


 Procedure GetPort7( var A:Byte);
 begin asm  mov dx,0331h; in al,dx; les bx, A; mov es:[bx],al end end;

 Procedure GetPort6( var A:Byte);
 begin asm  mov dx,0301h; in al,dx; les bx, A; mov es:[bx],al end end;




(*
  Procedure GetPortC( var A:Byte);
  begin asm  mov dx,019Ch; in al,dx; les bx, A; mov es:[bx],al end end;
*)
  Procedure GetPortB( var A:Byte);  {Switches}
  begin asm  mov dx,0331h; in al,dx; les bx, A; mov es:[bx],al end end;



 Procedure SetPort6_1( var A:Byte); {Fire Pulse}
 begin asm   mov dx,0301h; les bx, A; mov al,es:[bx];  { (A) > al}
             or al,bit1;   mov es:[bx],al;  out dx,al end end;



 Procedure ResetPort6_1( var A:Byte);  {Fire Pulse}
 begin asm  mov dx,0301h; les bx, A; mov al,es:[bx];
     and al,255-bit1; mov es:[bx],al; out dx,al end  end;
```

```
Procedure SetPort6_7( var A:Byte); {A/D Board}
begin asm  mov dx,0335h; les bx,A; mov al,es:[bx];
     or al,bit7;  mov es:[bx],al; out dx, al end end;


Procedure ResetPort6_7( var A:Byte);
begin asm  mov dx,0335h; les bx,A; mov al,es:[bx];
     and al,255-bit7; mov es:[bx],al; out dx,al end end;


Procedure SetPort6_0(var A:Byte);
begin asm  mov dx,0301h; les bx,A; mov al,es:[bx]; { (A) > al}
  or al,bit0; mov es:[bx],al; out dx,al end end;



Procedure ResetPort6_0(var A:Byte);  {Timing Pulse}
begin asm  mov dx,0301h; les bx, A; mov al,es:[bx];
  and al,255 - bit0; mov es:[bx],al; out dx,al end end;



Procedure Convert(var s:real; var nn:i8);
begin
{real: s:1   f:39    e:8. v := (-1)**s*2**(e-129)*(1.f).  if e=0 then v:=0}
{     b47  b46-b8  b7-b0  }
   asm          {MSByte  ah, al, ch, cl  LSByte}

   les bx,nn
   mov al,es:[bx+4]    { al: s.in m.2 m.1 m.0  x   x  b25  b24}
   and al,3          { al:  0  0  0  0  0  0  b25  b24}

   mov dl,es:[bx+5]    { dh: s.in m.2 m.1 m.0  x  x  b27  b26}
   and dl,3          { dh:  0  0  0  0  0  0 b27  b26}
   rol dl,2
   or al,dl          { al:  0  0  0  0 b27 b26 b25 b24}

   mov dl,es:[bx+6]    { dl: s.in m.2 m.1 m.0  x  x  b29  b28}
   and dl,3
   rol dl,4
   or al,dl

   mov dl,es:[bx+7]    { dl: s.in m.2 m.1 m.0  x  x  b31  b30}
   and dl,3
   rol dl,6
```

```
    or al,dl          {al:  b31 b30 b29 b28 b27 b26 b25 b24}

    mov cl,es:[bx+2]   {cl: b23 b22 b21 b20 b19 b18 b17 b16}
    mov dh,es:[bx+1]   {dh: b15 b14 b13 b12 b11 b10  b9  b8}
    mov dl,es:[bx+0]   {dl:  b7  b6  b5  b4  b3  b2  b1  b0}
    mov ch,0          {ch:  0  0  0  0  0  0  0  0}
    test al,128       {al: b31  0  0  0  0  0  0  0}
    jz @00            {Jump if negative}

       {sign:1, al:8, cl:8, dh:8, dl:8, 00:8, ah:8}
    xor al,255    {b31 b30 b29 b28 b27 b26 b25 b24}
    xor cl,255    {b23 b22 b21 b20 b19 b18 b17 b16}
    xor dh,255    {b15 b14 b13 b12 b11 b10 b09 b08}
    xor dl,255    {b07 b06 b05 b04 b03 b02 b01 b00}
    add dl,1;   adc dh,0;    adc cl,0;    adc al,0;    mov ch,128

  @00:mov ah,128+32   {sign:1, al:8, cl:8, dh:8, dl:8, 00:8, ah:8}
    test al,255;      {Check for all zeros}  jnz @3

    mov al,cl ; mov cl,dh ; mov dh,dl ; mov dl,0 ; mov ah,128+8+8+8
    test al,255; jnz @3

    mov al,cl ; mov cl,dh ; mov dh,00 ;          mov ah,128+8+8
    test al,255; jnz @3

    mov al,cl ; mov cl,00 ;                  mov ah,128+8
    test al,255; jnz @3

    mov al,00 ;                          ; mov ah,0
    jmp @57         {Finished}

   @3:                  test al,128;     jnz @57
     dec ah;            test al,64;      jnz @56
     dec ah;            test al,32;      jnz @55
     dec ah;            test al,16;      jnz @54
     dec ah;            test al,8;       jnz @53
     dec ah;            test al,4;       jnz @52
     dec ah;            test al,2;       jnz @51
     dec ah
;              {al, cl, dh, dl, 00, ah}
      rcl dl,1; rcl dh,1; rcl cl,1; rcl al,1
   @51: rcl dl,1; rcl dh,1; rcl cl,1; rcl al,1
   @52: rcl dl,1; rcl dh,1; rcl cl,1; rcl al,1
   @53: rcl dl,1; rcl dh,1; rcl cl,1; rcl al,1
   @54: rcl dl,1; rcl dh,1; rcl cl,1; rcl al,1
```

```
  @55: rcl dl,1; rcl dh,1; rcl cl,1; rcl al,1
  @56: rcl dl,1; rcl dh,1; rcl cl,1; rcl al,1
  @57: and al,127; or al,ch; les bx,s;          mov es:[bx+5],al
     mov es:[bx+4],cl;    mov es:[bx+3],dh;   mov es:[bx+2],dl
     xor al,al;          mov es:[bx+1],al;   mov es:[bx+0],ah
  end; {Asm}
end; {Proc Convert}



     Procedure Ack_Lo(var CNT:byte);    begin asm
     mov dx,0334h; mov ah,0
  @0: dec ah; jz @1; in al,dx;    test al,128 {bit7};   jnz @0
  @1: les bx,CNT;    mov es:[bx],ah end end;


     Procedure Ack_Hi(var CNT:byte);
     begin asm
     mov dx,0334h;  mov ah,0
  @0: dec ah; jz @1;  in al,dx;    test al,128 {bit7};    jz @0
  @1: les bx,CNT;    mov es:[bx],ah  end end;


     Procedure Get_Result(var a1,a0:byte);
     begin asm   mov dx,0336h; les bx,a0;  in al,dx;  mov es:[bx],al
     mov dx,0334h;  les bx,a1;  in al,dx;  mov es:[bx],al end end;



     Procedure Ext_Sign_Bit(var a0:byte);
     begin asm  les bx,a0; mov al,es:[bx];
     and al,15; test al,8; jz @0; or  al, 240
  @0: mov es:[bx],al end end;



  Procedure Int(var n2:byte; var n4,n3:byte);
  begin asm   les bx,n2; mov al,es:[bx]; mov ah,al;
  and al,112 {Mask channel};ror al,4; les bx,n4; mov es:[bx],al;mov al,ah;
  and al, 12 {Mask set}; ror al,2;  les bx,n3; mov es:[bx],al end end;



  Procedure C16(var count:word; var n0,n1:byte);
```

```
begin asm   les bx,n0; mov al,es:[bx]; les bx,n1; mov ah,es:[bx]
les bx,count; mov es:[bx],ax end end;


Procedure get(var n0:byte);
begin asm  mov dx,0336h;in al,dx;les bx,n0;mov es:[bx],al end end;


Procedure GetReading (var nn:I88; var Count:Word; var n4,n5 :Byte );
var j,k,n3,n2,n1,n0,CNT :Byte;
begin {l 1}
error := false;
j:= 0;
n5:=0;          n2 := 0;
while  (j < 7) do
 begin   {j 2}
    inc(n5);

    k := 0;
    n4 := 0;
    while (k<3) do
      begin  {k 3}
       inc (n4);
       if (( n2 and 128) = 0) then
         begin   {0 4}
            setport6_7(Port6);
            ack_hi(CNT);          {Wait for ack to go high}
       {     if (CNT =0) then  exit;}
            get_result (n2,n0);
            int(n2,j,k);
            nn[j,k+4] := n2;
            nn[j,k  ] := n0
         end    {0 4}
      else
         begin   {1 4}
            resetPort6_7(Port6);
            ack_lo(CNT);          {Wait for ack to go low}
       {    if (CNT =0) then exit;  }
            get_result (n2,n1);
            int(n2,j,k);
            nn[j,k+4] := n2;
            nn[j,k  ] := n1 ;
         end;   {1  4}
```

```
        end;  {k 3}

{if n4 <> 4 then error := true;}

  end;   {j 2}
  setPort6_7(Port6);    {*}
  resetPort6_7(Port6);

  if n5 <> 8 then error := true;

  c16(count,nn[6,3],nn[7,3]);

end;

    Procedure ReadShaftEncoder_Azi (var S:seal;var P:byte; var T_Word:word);
    var a,b:byte;
    begin asm     {Do not change bits P.1 and P.0}
    les bx,P; mov ah,es:[bx]; and ah, 03h {ah:0000 00nn}
;
    mov dx,301h;
    mov al,ah; or al,bit2; out dx,al   {al:0000 01nn} {Set Update Command}


    {Delay for 1 us}
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;

    les bx,t_word;
    mov al,ah;  out dx,al {al:0000 00nn} {Reset Update Command}
;
    mov dx,331h
 @0: in al,dx     ;    and al, bit6 ;   jz @0;  {Wait for update complete}
;
    mov dx,301h            {Set address 0}
    mov al,ah ; or al, bit4   ;out dx,al {al:0001 00nn}

    {Delay for 1 us}
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
```

```
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    mov dx,332h  ;   in al,dx ;   mov es:[bx],al; {Read LSByte }
;
    mov dx, 301h                    {ReSet address 0}
    mov al,ah  ; out dx,al   {al:0000 00nn}
                               {Set address 1}
            or al,bit3  ;    out dx,al  {al:0000 10nn}

    {Delay for 1 us}
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    mov dx,332h  ;   in al,dx ;   mov es:[bx+1], al ; {Read MSByte }

    mov dx, 301h                    {Reset Address 1}
    mov al,ah    ;   out dx,al     {al:0000 00nn}

  end; {asm}
 { a:=(t_word div 256);  b:= (t_word-256*a);
  writeln('azi ' ,t_word:12,a:12,b:12);
 }  S := t_word ;
  end; {procedure}


     Procedure ReadShaftEncoder_Ele (var S:seal;var P:byte; var T_Word:word);
    var a,b:byte;
    begin asm    {Do not change bits P.1 and P.0}
    les bx,P     ; mov ah,es:[bx];     and ah, 03h {ah:0000 00nn}
;
    mov dx,301h;
    mov al,ah; or al,bit5; out dx,al {al:0010 00nn} {Set Update Command}

    {Delay for 1 us}
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
```

```
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    les bx,t_word;
    mov al,ah;   out dx,al {al:0000 00nn} {Reset Update Command}
;
    mov dx,331h
  @0: in al,dx    ;   and al, bit7 ;   jz @0;  {Wait for update complete}
;
    mov dx,301h                         {Set address 0}
    mov al,ah    ;   or al, bit7  ;   out dx,al {al:1000 00nn}

    {Delay for 1 us}
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    mov dx,330h  ;   in al,dx ;   mov es:[bx],al ; {Read LSByte}
;
    mov dx, 301h                    {ReSet address 0}
    mov al,ah    ;   out dx,al               {al:0000 00nn}

                               {Set address 1}
                 or al,bit6  ;   out dx,al  {al:0100 00nn}

    {Delay for 1 us}
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    mov dx,330h  ;   in al,dx ;   mov es:[bx+1], al ; {Read MSByte}

    mov dx, 301h                    {Reset Address 1}
    mov al,ah    ;   out dx,al   {al:0000 00nn}
```

```
 end; {asm}
{ a:=(t_word div 256);   b:= (t_word-256*a);
}
 S := t_word ;


end; {procedure}


 Procedure Wheel_Pulse;
 begin asm mov dx,0306h;
         mov al,0;     out dx,al;
         mov al,bit0;  out dx,al
         end end;



 Procedure Wheel_Read(var Port9_In:Byte);
 begin asm  les bx,Port9_In; mov dx,0305h;
    in al,dx; mov es:[bx],al end end;


 Procedure Sensors (var wy,wz,wx,joy_p,joy_o,  c_azi, c_ele  :seal;
                    var Count:word);
 var j,k:byte;
 begin
    GetReading (     nn, Count, n4,n5    );
    j := 3; if (error=false) then begin
    for k := 0 to 7 do nnn[k] := nn[j,k]; Convert(zzzz,nnn) end;wy:=- zzzz;

    j  := 7;if (error=false) then begin
    for k := 0 to 7 do nnn[k] := nn[j,k]; Convert(zzzz,nnn) end;wz:= - zzzz;

    j := 6; if (error=false) then begin
    for k := 0 to 7 do nnn[k] := nn[j,k]; Convert(zzzz,nnn) end;wx:=  zzzz;

    j := 2; if (error=false) then begin
    for k:=0 to 7 do nnn[k]:=nn[j,k]; Convert(zzzz,nnn) end;joy_P :=-zzzz;



    j := 4; if (error=false) then begin
    for k:=0 to 7 do nnn[k]:=nn[j,k]; Convert(zzzz,nnn) end;c_azi := zzzz;
```

```
    j := 0; if (error=false) then begin
    for k:=0 to 7 do nnn[k]:=nn[j,k]; Convert(zzzz,nnn) end;c_ele := zzzz;



    j := 5;if (error=false) then begin
    for k:=0 to 7 do nnn[k]:=nn[j,k]; Convert(zzzz,nnn) end;joy_O :=-zzzz;
end;

Procedure Draw (a,b:integer; var d,e:integer);
var c:boolean;
begin                    a:=d; b:=e;
      {a,b Corner reference,   0 to 511}
 if a<20 then a:=20; if b<20 then b:=20;
 if a>480 then a:=480; if b>480 then b:=480;

 G_Set(a,b,App,Bpp);
 ApP := a; BpP:= b;

end;

 Procedure Fit(var ZZZ,ZZ,Z,YZ,SZ:seal);
 var Y9:seal;
 begin YZ :=(5*z+zz+zz-zzz)/6; y9 := (z+zz+zzz)/3; SZ := (YZ - Y9) end;



(* Procedure GetPort77(var a,b:byte);
  begin
  a:=0; b:=255;
  while a<>b do
  begin asm mov dx,0331h; les bx,b; in al,dx; mov es:[bx],al;
     les bx,a; in al,dx; mov es:[bx],al end end end;

 *)

 Procedure Encoders(var SaeO, SaeP:Seal);
 begin
  ReadShaftEncoder_Ele( SaeO, Port6,temp_word);  {0 <= SaeO <= 65535}
  ReadShaftEncoder_Azi( SaeP, Port6,temp_word);  {0 <= SaeP <= 65535}
  saeo:= saeo + saeo_c; saep := saep + saep_c;
  if saeO>32767 then saeo:=saeo-65535; if saep>32767 then saep:=saep-65535;
```

```
   SaeO :=  SaeO*ConversionToRadians;  SaeP := SaeP*ConversionToRadians ;
{gotoxy(1,1);     writeln('saeo saep ',saeo*180/pi:12:3,saep*180/pi:12:3)};
   end;

    (*
   Procedure GetPortD(var value,sam:byte);
   begin sam:=0;value:=255;while sam <> value do begin asm mov dx,019dh;
   les bx,value;in al,dx; mov es:[bx],al;les bx,sam;in al,dx; mov es:[bx],al
   end end end;
*)

(*  Procedure PutPortC(nnnn:byte;var value:byte); begin value:=nnnn;
  asm  mov dx,019ch; les bx,value; mov al,es:[bx]; out dx,al end; end;
*)

(*   Procedure BitGet(var a,b:byte);
   begin asm
   les bx,B; mov al, es:[bx]; and al,40h; add al,al; mov ah,al
   les bx,A; mov al, es:[bx];  or al,ah;  mov es:[bx],al end; end;
 *)


   Procedure SerialToReal(var s:ar128b;var v:real);
   begin
   asm  les bx,S; mov ch,es:[bx  ]; mov cl,es:[bx+1]; mov dh,es:[bx+2];
   mov dl,es:[bx+3]; mov ah,es:[bx+4];mov al,es:[bx+5];
   les bx,V; mov es:[bx  ],ch; mov es:[bx+1],cl; mov es:[bx+2],dh;
   mov es:[bx+3],dl; mov es:[bx+4],ah;mov es:[bx+5],al;end; end;



   Procedure ConvertSing(n:word; var s:ar128b; var x:real);
   var m,k:word; t:ar128b;
   begin m:=1; for k := n to n+1 do begin t[m]:=s[k+k-1];inc(m);t[m]:=s[k+k];
   inc(m); end; t[6]:=t[4];t[5]:=t[3]; t[4]:=t[2];t[3]:=0; t[2]:=0;
   SerialToReal(t,x); end;



   Procedure ConvertReal(n:word; var s:ar128b;var x:real);
   var m,k:word; t:ar128b;
   begin m:= 1; for k := n to n+2 do
       begin t[m] := s[k+k-1]; inc(m); t[m] := s[k+k]; inc(m);
   end;
   serialToReal(t,x)
```

```
end;


Procedure PutPort6(var a:byte);
begin asm  mov dx,0301h;les bx,a; mov al,es:[bx]; out dx,al; end end;


Procedure SuperElevation(var Range, Angle:Seal);
{Units are meters and radians}
begin
Angle := -1.00358e-4 + range*(7.34523e-6 + range*(-3.81355e-10 + range*2.49212e-12)) end;
(*

Procedure LaserRangeFinder;
begin
        begin
            Port6 := Port6 and (255-32);
            PutPort6(Port6);
            Delay_(200);
            Port6 := Port6 or 32;
            PutPort6(Port6);
;
            Port7Bit6 :=0;
            while Port7Bit6 = 0 do
                  begin
                      GetPort7(Port7In);
                      Port7Bit6 := Port7In and 64;
                      if keypressed=true then Port7Bit6:=1;
                  end;
                GetPortC(PortC0);
                port6 := port6  and 255 -  8;
                PutPort6( Port6 );
                delay_(400);
                GetPortC(PortC1);
                Port6 := port6  or 8;
                dist := 0;
                if portc0 and 128 > 0 then dist:= dist + 200;
                if portc0 and  64 > 0 then dist:= dist + 2000;
                if portc0 and  32 > 0 then dist:= dist + 4000;
                if portc0 and  16 > 0 then dist:= dist + 8000;
                if portc0 and   4 > 0 then dist:= dist + 400;
                if portc0 and   2 > 0 then dist:= dist + 800;
                if portc0 and   1 > 0 then dist:= dist + 1000;
```

```
                if portc1 and  64 > 0 then dist:= dist + 20;
                if portc1 and  32 > 0 then dist:= dist + 40;
                if portc1 and  16 > 0 then dist:= dist + 80;
                if portc1 and   8 > 0 then dist:= dist + 100;
                if portc1 and   2 > 0 then dist:= dist + 5;
                if portc1 and   1 > 0 then dist:= dist + 10;
                range := dist + 0.1;
                if range < 5 then range := 5;
        end; {Reading the Laser Range Finder}
  end;
 *)

  Procedure Time_Of_Flight(var a, b: seal);
  begin a:= -7.002e-3 + b*(1.1388303e-3 + b*(2.23335e-7 + b*1.4562e-10)) end; {b denotes the
range}


  Procedure MatrixIntegrate; begin
  Mat_Der( xi, et, ze, ch,   wx, wy, wz,   xidp, etdp, zedp, chdp);
  Integrate(xi,et,ze,ch,  xid0,xidp, etd0,etdp, zed0,zedp, chd0,chdp);
  Up_Date(xid0, xidp,  etd0, etdp,  zed0, zedp,  chd0, chdp);
  Normalize(xi, et, ze, ch);  {Assures orthonormality}
  if WC then Wheel_Pulse;  {Allows time for the embedded controller}
  Mat( xi, et, ze, ch, A); {Defines the A transformation matrix} end;


  Procedure DriftCorrection; begin wx := Coeff1 * (wx/count - DriftX);
  wy:=Coeff1*(wy/count - DriftY);  wz := Coeff1 * (wz/count - DriftZ); end;


  Procedure DAC00(var xx:integer);
  CONST PORT = $00;       BASE = $0f3c0;
  begin asm
      les bx,xx; mov ax,es:[bx];                  {RUBY-MM, Port = 00h}
      xor ah,8h;
      mov dx,BASE + 08h;  out dx, al;  {LS8Bs -> BASE + 08h, same for all}
      mov dx,BASE +PORT;  mov al,ah; out dx, al;    {MS4Bs -> BASE + Port}
  end

  end;

  Procedure DAC01(var xx:integer);
  CONST PORT = $01;       BASE = $0f3c0;
  begin asm
```

```
    les bx,xx; mov ax,es:[bx];                {RUBY-MM, Port = 00h}
    xor ah,8h;
    mov dx,BASE + 08h;  out dx, al;  {LS8Bs -> BASE + 08h, same for all}
    mov dx,BASE +PORT;  mov al,ah; out dx, al;    {MS4Bs -> BASE + Port}
end

end;

Procedure DAC02(var xx:integer);
CONST PORT= $02;       BASE = $0f3c0;
begin asm
    les bx,xx; mov ax,es:[bx];                {RUBY-MM, Port = 00h}
    xor ah,8h;
    mov dx,BASE + 08h;  out dx, al;  {LS8Bs -> BASE + 08h, same for all}
    mov dx,BASE +PORT;  mov al,ah; out dx, al;    {MS4Bs -> BASE + Port}
end

end;

Procedure DAC03(var xx:integer);
CONST PORT= $03;       BASE = $0f3c0;
begin asm
    les bx,xx; mov ax,es:[bx];                {RUBY-MM, Port = 00h}
    xor ah,8h;
    mov dx,BASE + 08h;  out dx, al;  {LS8Bs -> BASE + 08h, same for all}
    mov dx,BASE +PORT;  mov al,ah; out dx, al;    {MS4Bs -> BASE + Port}
end

end;

Procedure DAC_UPDATE;
CONST BASE = $0f3c0;
begin asm
    mov dx,BASE + 09h;  in al,dx; {Update all 8 channels simultaneously}
end end;

Procedure GetR6(Var R6:byte);
begin asm  mov dx,0236h; les bx,R6; in al,dx; mov es:[bx],al end end;

Procedure Vertical_Blank(var R6P,R6:byte);
begin
    while  not  ( (R6 = Bit3) and (R6P=0) ) do
    begin R6P := R6;  GetR6(R6); R6 := R6 and Bit3  end; {Synchronization}
end;
```

```
Procedure CLR( a:byte);   begin putp6(a); end;
Procedure STT( a:byte); var b:byte; begin b:=a+1;   putp6(b); end;

Procedure DELAYS(time:word);
var hr0,min0,sec0,hun0,i:word;
begin
    GetTime(hr,min,sec,hun);
    hun0:=hun; while hun0=hun do GetTime(hr,min,sec,hun);
   for i:=1 to time do begin
   hun0:=hun; while hun0=hun do GetTime(hr,min,sec,hun);
    end;end;

Procedure RESET_CX;
const IN_LUT_ENABLE=$14; OUT_LUT_ENABLE=$24;
BEGIN
    asm  mov dx,0236h; in al,dx; {Read port 6 to wake the board up} end;
    Delays(1);
    clr(IN_LUT_ENABLE);
    clr(OUT_LUT_ENABLE);
    P6:= $59; PutP3(P6); P6:=$15; PutP6(P6); P6:=$2f; PutP6(P6);{Reboot}
    Delays(1); {Delay 55 to 100 ms to finish}
    P6:=$47; PutP3(P6); {Put board in the CX100 mode}
    Delays(1);  {Delay 55 to 100 ms to finish}
END;

Procedure RAM_ON; begin Put6($1b,P6) end;

Procedure RAM_OFF; begin Put6($1a,P6) end;


Procedure Frame;
var n,h,i:word;
begin
    m:=0; n:=$00ff;   lll:=1; {high order byte of "n" denotes color}
    for i:= 0 to 511 do
    begin h:=0;  set_pix(i,h,lll,n); h:=255; set_pix(i,h,lll,n); end;
end;


Procedure Bright( vv,hh:word; var v0,h0,v1,h1:integer; var b1:byte);
Const  Size=78 ;  SizeR=511-size; Size2=Size div 2; Size3 = Size div 3;

begin
 v0 := vv - size2; h0 := hh - size2;
 {Convert v0 and h0 to edge reference}
```

```
  if v0>SizeR then v0:=SizeR; if h0>SizeR then h0:=SizeR;

  asm
  cli
  les bx,H0   {h0, 0 to 511}
  mov cx,es:[bx] {cx: 0 0 0 0  0 0 0 b8  b7 b6 b5 b4  b3 b2 b1 b0, H}
  les bx,V0   {v0, 0 to 511}
  mov dx,es:[bx] {dx: 0 0 0 0  0 0 0 a8  a7 a6 a5 a4  a3 a2 a1 a0, V}
  rol dx,1     {dx: 0 0 0 0  0 0 a8 a7  a6 a5 a4 a3  a2 a1 a0 0}
  xor al,al    {ax: x x x x  x x x x  0 0 0 0  0 0 0 0}
  mov ah,dl    {ax:a6 a5 a4 a3  a2 a1 a0 0  0 0 0 0  0 0 0 0}
  add cx,ax    {cx:a6 a5 a4 a3  a2 a1 a0 b8  b7 b6 b5 b4  b3 b2 b1 b0,cx}
  mov bx,cx    {bx:a6 a5 a4 a3  a2 a1 a0 b8  b7 b6 b5 b4  b3 b2 b1 b0,bx}
  mov al,dh    {ax:a6 a5 a4 a3  a2 a1 a0 0  0 0 0 0  0 0 a8 a7,ax}
  and al,03h   {ax:a6 a5 a4 a3  a2 a1 a0 0  0 0 0 0  0 0 a8 a7,ax}
  mov dx,236h  {dx: 0 0 0 0  0 0 1 0  0 0 1 1  0 1 1 0,dx}
  out dx,al    {ax:a6 a5 a4 a3  a2 a1 a0 0  0 0 0 0  0 0 a8 a7,ax}
  mov dx,0a000h {dx: 1 1 0 1  0 0 0 0  0 0 0 0  0 0 0 0,dx}
  mov es,dx    {es: 1 1 0 1  0 0 0 0  0 0 0 0  0 0 0 0,es}
  xor ah,ah    {ax: 0 0 0 0  0 0 0 0  0 0 0 0  0 0 a8 a7,ax}
  mov di,ax    {di: 0 0 0 0  0 0 0 0  0 0 0 0  0 0 a8 a7,di}
  mov dh,size3; {dx: 0 0 0 1  0 1 0 0  0 0 0 0  0 0 0 0,dx}
  push dx
@2: pop dx
  cmp dh,0;
  je @10
  dec dh
  push dx
  mov dl,size3;

@3:
{   mov dh,255;
  mov es:[bx],dh;

 }
  mov dh, es:[bx]
  add ah,128; add dh,128
  cmp ah,dh
  jg @4      {Jump if ah >= video ram}

  {Get here if video ram > ah}  {Vertical                Horizontal}
@5: add dh,128; add ah,128
  mov cx,bx    {cx:a6 a5 a4 a3  a2 a1 a0 b8  b7 b6 b5 b4  b3 b2 b1 b0,cx}
  mov ah,dh    {ax:v7 v6 v5 v4  v3 v2 v1 v0  0 0 0 0  0 0 a8 a7,ah}
  mov di,ax    {di: n n n n  n n n n  0 0 0 0  0 0 a8 a7,di}
```

```
   add ah,128; add dh,128

@4: add ah,128; add dh,128
  add bx,3
   dec dl
   jnz @3

@6: add bx, 3 * 512 - size  ;
   jnc @2
   inc al
   push dx
   mov dx, 236h
   out dx,al {Update page select bits}
   pop dx
   jmp @2

@10: mov ax,di    {ax:v7 v6 v5 v4  v3 v2 v1 v0   0 0 0 0   0 0 a8 a7,ax}
            {cx:a6 a5 a4 a3  a2 a1 a0 b8  b7 b6 b5 b4  b3 b2 b1 b0,cx}
   mov dx,cx ; and dx,01ffh;    les bx,h1 ; mov es:[bx],dx
   les bx,b1 ; mov es:[bx],ah ; {brightness to b1}
   mov ah,al       {ax: 0 0 0 0   0 0 a8 a7   0 0 0 0   0 0 a8 a7}
               {cx:a6 a5 a4 a3  a2 a1 a0 b8  b7 b6 b5 b4  b3 b2 b1 b0}
   mov al,ch       {ax: 0 0 0 0   0 0 a8 a7  a6 a5 a4 a3  a2 a1 a0 b8}
   clc     ; {Clear the carry flag.}
   rcr ax,1        {ax: 0 0 0 0   0 0 0 a8  a7 a6 a5 a4  a3 a2 a1 a0}
   les bx,v1; mov es:[bx], ax
   sti
   end; {asm} {v1 and h1 0 to 511, edge reference}
   end;
```

{ The origin for the Body Fixed Coordinates (BFCs) is the sight
aligned with the orientation of the buggy.
xx, yy and zz define the location of the sight in Inertial Coordinates (ICs).
xw, yx and zw define the location of the counter wheel in ICs.
xb, yb and zb define the location of the target wrt the BFCs.
xa, ya and za define the location of the target wrt the ICs.}

```
(*--------------------------------MAIN-----------------------------------*)
const IN_LUT_ENABLE=$14; OUT_LUT_ENABLE=$24;
DISPLAY_RAM=$08;RAM_ENABLE=$1A;
```

```
BEGIN {main}        clrscr;

                    R6p := 0; R6:=BIT3;

                    RESET_CX;

                    RAM_On;

                    Vertical_Blank(R6P,R6); {Wait for vertical blank}

{ Disp_Ovl}         P6:=$27; PutP6(P6); d; { $27 overlay , $26 no overlay}

{ Ovr_Ena}          P6:=$28; PutP6(P6); d; { $28 video ram, $29  overlay ram}

{ Interrupt Enable} P6:=$10; PutP6(P6); d;

{Trigger Enable}    Put6($12,P6);      d;
{ High_Res}         P6:=$0a; PutP6(P6); d;

{ Display_Ram}      P6:=$09; PutP6(P6); d; { $08 live, $09 ram}
                    clr(Out_LUT_Enable);d;

{Blank Display}     Put6($16,P6);      d;  { $16 display  $17  blank}

{Acquire Request}   Put6($06,P6);      d;

                    RAM_ON; d;

{Overlay Ram Select} P6:=$29; PutP6(P6);d; { $29 }
   Frame;           {Rewrites the overlay ram, all pixels  }
{Video Ram Select}  P6:=$28; PutP6(P6);d; { $28}

                    Put6($08,P6);
   Frame;           {Rewrites the video ram, all pixels  }

   R6P := R6;
   while  not  ( (R6 = Bit3) and (R6P=0) ) do
   begin R6P := R6;  GetR6(R6); R6 := R6 and Bit3  end; {Synchronization}

  P6 := $27; PutP6(P6);  { $27 Overlay on}  d;
  P6 := $29; PutP6(P6); { $29 Select the overlay memory} d;
   vc:=256;   hc:=256;                      {Edge coordinates}

   Draw (vvvv,hhhh,vc,hc);
```

```
    P6 := $28; PutP6(P6); { $28 Select the video memory} d;

    R6p := R6;
    while  not  ( (R6 = Bit3) and (R6P=0) ) do
    begin R6P := R6;  GetR6(R6); R6 := R6 and Bit3  end; {Synchronization}
    GetR6(P6);P6:=P6 and Bit3;

  G_Init; {Initializes the video display board}

  app:=0;bpp:=0; {Initializes the graphics}
  Port6 :=125; ResetPort6_1(Port6);
  GetPort7(P7In);

{while true do begin   setport6_7(port6) ; d; resetport6_7(port6); d;  end;}
GPS:=FALSE; WC:=true;{ if((P7In and 4)=0) then GPS := TRUE else WC := TRUE;}

    ao := 0; ap := 0; {Dummy variables used by the stepper procedure}

    JoyO_:=0; JoyP_:=0; {Used in the joystick integration mode}

    C_Joy_O:=pi/(180*131072);C_Joy_P:=pi/(180*131072);{Joy stick sensitivity}

    Flop:=1; SaeO_:= 0; SaeP_ := 0; OO:=0;  O:=0;  PP:=0;  P:=0;

    sensors(wy,wz,wx,joyp,joyo,c_azi, c_ele, Count);  {Done for synchronization}

    c_a_drift := c_azi/count;          c_e_drift :=  c_ele/count;

    get_result (n2,n0);

    Ack_Lo(CNT); {Test slave's output - low normal state}
    if (CNT = 0) then
     begin
       writeln( 'Slave in wrong state.  Press any key');
       halt;
     end;
   { writeln('sensors ');}

    sum1 := 0; sum2 := 0; B1 := 0; B2 := 0;
;
  { ResetPort6_0(Port6);}   Fir := False;  LLL := 0;

    xc:=0;  dt2 :=dt/2;
    xid0:=0; etd0:=0; zed0:=0; chd0:=0;
    Speed:=0;  speed_:=0; wi:=65535;   {used in calculating vehicle speed}
```

```
Range := 100; {Center of Screen (COS) Sight to Target distance}
{xb, yb, zb define the target in Body Fixed Coordinates (BFCs)}
Encoders (SaeO,SaeP); {Center of the Screen} {O elevation,  P azimuth}
CosP := cos(SaeP); SinP := sin(SaeP);CosO := cos(SaeO); SinO := sin(SaeO);
xb:=CosO*CosP*Range;
yb:=CosO*SinP*Range;
zb:=SinO*Range;{COS, wrt BFCs, BFCs}

if WC then
 begin
  Wheel_Pulse; {Initializes wheel pulse counter}
  xx:=0;yy:=0;zz:=0; {xx yy zz:location of the origin of the BFCs, ICs}
 end;


Initialize_Q(xi,et,ze,ch);  {0,0,0,1}
Mat( xi, et, ze, ch, A); {Defines the A transformation matrix, A=[1]}

{Initial position of the Reference Wheel, ICs} {A=[1]}
xw:=xx - WheelToSight_X; yw:=yy - WheelToSight_Y; zw:=zz-WheelToSight_Z;
{xw yw zw: the Reference Wheel wrt BFCs, ICs}

SuperElevation(Range, SE); {SE defines the super elevation angle}
  ZSE := SE*Range; {ZSE becomes the apparent z offset of the target}

BFCsToICs(xa,ya,za,xb,yb,zb); {Target's Position, COS, wrt BFCs, ICs}
xxx:=xa+xx; yyy:=ya+yy; zzz:=za+zz; {COS&Range, wrt ICs,ICs}
IntP:=0; IntO:=0; yo:=0; yp:=0; so:=0; sp:=0;

writeln(' xx  yy  zz ', xx:12:2, yy:12:2, zz:12:2);
writeln(' xb  yb  zb ', xb:12:2, yb:12:2, zb:12:2);
writeln(' xa  ya  za ', xa:12:2, ya:12:2, za:12:2);
writeln('xxx yyy zzz ', xxx:12:2, yyy:12:2, zzz:12:2);
Looper := 0; {Used by the wheel counter to automatically go to drift mode}
writeln('Entering master loop');
(*
for lll:= 1 to 40 do
begin
 sensors(wy,wz,wx,joyp,joyo,c_ele, c_azi, Count);  {Done for synchronization}
 c_a_drift := c_azi/count;
 c_e_drift :=  c_ele/count;
 driftx:=wx/count; drifty:=wy/count; driftz:=wz/count;
 c_a_drift := c_a_drift + ( c_azi - c_a_drift)*0.001;
 c_e_drift := c_e_drift + ( c_ele - c_e_drift)*0.001;
 driftx:=driftx+(wx-driftx)*0.001;
```

```
    drifty:=drifty+(wy-drifty)*0.001;
    driftz:=driftz+(wz-driftz)*0.001;

  end;
*)
  sensors(wy,wz,wx,joyp,joyo,c_ele, c_azi, Count);  {Done for synchronization}
  driftx:= wx/count; drifty:=wy/count; driftz:=wz/count;
  c_azi:=c_azi/count; c_ele:=c_ele/count;

  driftx:=160.0; drifty:=257.0; driftz:=-377.0;
  c_a_drift:=2157; c_e_drift:=574;

vxx := 0; vyy := 0; vzz:=0; {Speed of the target in BFCs.}

t:=0;
lp :=0 ; {byte, Loop Counter}

too:=0;tpp:=0; {Used in stepper driver}    v_d:=0; h_d:=0;
clrscr;     V_Initial:=339; H_Initial:=355; vc:=V_Initial; hc:=H_Initial;


  for i := 0 to  6 do begin eeg[i]:=0;  eef[i]:=0  end;
  for efp:= 0 to 42 do begin eg[efp]:=0;  ef[efp]:=0 end;
  efp := 0; {Error File Pointer}
  s0:=0; s1:=0; s2:=0; s3:=0; s4:=0;
  for i := 0 to 6 do
    begin   s0:=s0+1; q:=i; s1:=s1 + q; q:=q*i; s2:=s2 + q; q:=q*i;
                    s3:=s3 + q; q:=q*i; s4:=s4 + q     end;
  aa[1,1]:=s2; aa[1,2]:=s1; aa[1,3]:=s2;
  aa[2,1]:=s1; aa[2,2]:=s0; aa[2,3]:=s1;
  aa[3,1]:=s2; aa[3,2]:=s1; aa[3,3]:=s0;
{
writeln(aa[1,1]:20:8,aa[1,2]:20:8,aa[1,3]:20:8);
writeln(aa[2,1]:20:8,aa[2,2]:20:8,aa[2,3]:20:8);
writeln(aa[3,1]:20:8,aa[3,2]:20:8,aa[3,3]:20:8);
 }  LuDcmp(2,aa,indx, d_temp);
{writeln(aa[1,1]:20:8,aa[1,2]:20:8,aa[1,3]:20:8);
writeln(aa[2,1]:20:8,aa[2,2]:20:8,aa[2,3]:20:8);
writeln(aa[3,1]:20:8,aa[3,2]:20:8,aa[3,3]:20:8);
 }
tempp:=0; writeln('b0810b');                printit:=false;

(* - - - - - - - - - - - - - Master Loop - - - - - - - - - - - - - - - - - *)

while true do
```

```
 begin
  if keypressed=true then
       begin writeln(driftx:16:6,drifty:16:6,driftz:16:6,
       c_a_drift:16:6,c_e_drift:16:6); halt end;


  GetPort7(P7In);

  Sensors(wy, wz,wx, JoyP, JoyO, c_ele, c_azi, Count);

{ if printit then
write('Sensors',wx/count:7:2,wy/count:7:2,wz/count:7:2,JoyP/count:7:2,JoyO/count:7:2,count:4);
}

  c_azi := c_azi/count;          c_ele := c_ele/count;
  c_azi := c_azi - c_a_drift;    c_ele := c_ele - c_e_drift;


  DriftCorrection;  {Cancels out static drift in wx, wy and wz}
{wx:=0;wy:=0;wz:=0; }
;  {Looper is used to turn on and off the drift corection}
  {and is based on when the last wheel pulse was detected.}

  if ((Looper=0) and (P7In and 1 = 0 )) then
   begin
    if wx < 0 then driftx := driftx - 0.2 else driftx := driftx + 0.2;
    if wy < 0 then drifty := drifty - 0.2 else drifty := drifty + 0.2;
    if wz < 0 then driftz := driftz - 0.2 else driftz := driftz + 0.2;
    end;
    if c_azi<0 then c_a_drift := c_a_drift - 1.2 else c_a_drift := c_a_drift + 1.2;
    if c_ele<0 then c_e_drift := c_e_drift - 1.2 else c_e_drift := c_e_drift + 1.2;

  MatrixIntegrate; {Update Quaterions based on angular rates}
;
  if WC then
   begin
     if wi<65535 then inc(wi) else speed :=0;
     Wheel_Read(Port9_In); {Number of pulses since last update}
     if (Port9_In = 0) then
        begin if wi > wi_ then speed_ := - wheel_click/(wi*dt) end
     else
        begin speed_:= - wheel_click/(wi*dt); wi_:=wi; wi:=0 end;
     { speed_:= - 6.7056;} {15mph in meters per second}
     Speed:= 0.99*Speed + 0.01*speed_;
     temp := speed*dt; {Speed is negative for rear firing}
```

```
      BFCsToICs(xx,yy,zz,WheelToSight_X,WheelToSIght_Y,WheelToSight_Z);
      xw := xw + Temp*a[1,1];xx:= xx + xw; {xw: Ref. Wheel's position, ICs}
      yw := yw + Temp*a[1,2];yy:= yy + yw; {xx: Scope's position, ICs}
      zw := zw + Temp*a[1,3];zz:= zz + zw;
      {xw, yw and zw: the Reference Wheel's position in IC}
      {xx, yy and zz: the location of the sight (origin of BFCs) in ICs}
      if Port9_In > 0 then looper:=1815 else if looper>0 then dec(looper);

    end;

    if printit then  write(' P9_In ', port9_In:4, ' ');
    if printit then  write(' wi_ ',wi_:4, ' ');
;
   Encoders (SaeO,SaeP); {Center of the Screen} {O elevation,  P azimuth}
   CosP := cos(SaeP); SinP := sin(SaeP);CosO := cos(SaeO); SinO := sin(SaeO);

   if PrintIt then write('saeo saep ',saeo:8:4,saep:8:4);


;

   {Routine for setting the Range}
   GetPortB(PortB_In);

   If (PortB_In and 4=0) then
   begin     {manual ranging and Target Position (BFCs) update}
     IntO:=C_Joy_O*JoyO/count; IntP :=-C_Joy_P*JoyP/count;
     xw:=0; yw:=0; zw:=0;
     If ( (PortB_In and 8) = 0 ) then  Range := Range + 0.1;
     If ( (PortB_In and 16) = 0 ) then  Range := Range - 0.1;
     if range < 5 then range := 5;
     xb:= CosO*CosP*Range ; yb :=  CosO*SinP*Range;  zb := SinO*Range;
     initialize_Q(xi,et,ze,ch);
     mat(xi,et,ze,ch,a);
     BFCsToICs(xa,ya,za,xb,yb,zb);{Center of Screen (COS) & Range, wrt BFCs}
     xxx:=xa+xx;yyy:=ya+yy;zzz:=za+zz;{COS&Range, wrt ICs,ICs}
     t:=0;

   end;

;
   {Target Acquire}

 if{((PortB_In and 4>0) and} (PortB_In and 16=0) then
    vyy:=4.47; {10 mph}
```

```
{    begin
     LaserRangeFinder;
     if range <10 then range:=100;
     IntO:= C_Joy_O*JoyO/count; IntP := -C_Joy_P*JoyP/count;
     xw:=0; yw:=0; zw:=0;
     xb := CosO*CosP*Range; yb :=  CosO*SinP*Range; zb := SinO*Range;
     BFCsToICs(xa,ya,za,xb,yb,zb);
     xxx:=xa+xx; yyy:=ya+yy; zzz:=za+zz;{COS&Range, wrt ICs,ICs}
    { sensors(we0,we1,we2,we3,we4,we5,we6, we7);

     t:=0;
   end;  }
   ;
   { t:=t + dt;}
    if (PortB_In and 8=0) then vyy:=0;
  xxx := xxx + vxx*dt; yyy := yyy+vyy*dt ; zzz := zzz+ vzz*dt;

{gotoxy(1,1);  writeln(t:12:2,xxx:12:2, yyy:12:2,zzz:12:2); }

   RRR:=XXX - XX; SSS:=YYY - YY; TTT:=ZZZ - ZZ; {COS&Range, wrt BFCs, ICs}
   Range:=sqrt(sqr(RRR) + sqr(SSS) + sqr(TTT)); {COS distance}
   ;
    Time_Of_Flight(TimeOfFlight,Range);
    {TimeOfFlight is an approximation neglecting the speed of the vehicle}
;
    ICsToBFCs(rr,ss,tt,rrr,sss,ttt);{COS&Range+Gravity, wrt BFCs (BFCs&ICs)}
    azi0 := atan(SS,RR); ele0 :=atan(TT,sqrt(sqr(RR)+sqr(SS))); {COS}
;
    SuperElevation(Range, SE); {SE defines the super elevation angle}
    ZSE := SE*Range; {ZSE becomes the apparent z offset of the target}
;
    dxx:=vxx*TimeOfFlight ; dyy:= vyy*TimeOfFlight; dzz:=vzz*TimeOfFlight;

    ICsToBFCs(dx,dy,dz,dxx,dyy,dzz);


    RR:=RR + Speed*TimeOfFlight - DX;   {RR SS TT: hit point, wrt BFCs, BFCs}
    SS := SS - DY;
    TT:=TT - ZSE - DZ ; {Gravity Drop of the Projectile}
    {COS&Range+Gravity, Lead Angle, wrt BFCs}
    azi1:=atan(SS,RR){+0.238/range}; ele1 :=atan(TT,sqrt(sqr(RR)+sqr(SS)));

    Slew := 0.002; if ((PortB_In and 8)=0) then slew := 0.007;
    JoyP := C_Joy_P*JoyP/count;
    if ((P7in and 2)=0) then IntP := IntP + JoyP*slew;
```

```
    JoyP_ := JoyP + IntP;
;
    Slew := 0.002; if ((PortB_In and 8)=0) then slew := 0.002;
    JoyO := - C_Joy_O*JoyO/count;
    if ((P7in and 2)=0) then IntO:=IntO + JoyO * slew {else t:=0};
    JoyO_ := JoyO + IntO;

if printit then write(' azi ',azi1:8:4,' speed ',speed:8:2);
;
   SaeO := SaeO + JoyO_; {Elevation} Saep := Saep + JoyP_; {Azimuth}
   D_El0 :=(SaeO - Ele0);          D_Az0 := (SaeP - Azi0);
   D_El1 :=(SaeO - Ele1);          D_Az1 := (SaeP - Azi1);
   {Elevation, O,  Azimuth, P}


   OOO := OO; OO:=O; O :=-D_El0;     PPP := PP; PP:=P; P := D_Az0;
   YO9:=YO; Fit(OOO,OO,O,YO,SO) ;  {Generates YO,SO}
   YP9 := YP; Fit(PPP,PP,P,YP,SP); {Generates YP,SP}
   {0O} Stepper_Driver(0,AO,TOO,YO,SO);

;              { * * * * * * * * * * }
 {gotoxy(1,1);}

 ef[efp]:=D_El0;  eg[efp]:=D_Az0;
 { ef[efp]:=tempp; eg[efp]:=tempp; tempp:=tempp+1;}

 ii:= efp; eef[0]:=ef[ii];
 for i :=1 to 6 do
   begin ii:=ii - 7; if ii<0 then ii := ii+43; eef[i]:=ef[ii] end;

 ii := efp; eeg[0]:=eg[ii];
 for i:=1 to 6 do begin ii:=ii-7;if ii<0 then ii:=ii+43; eeg[i]:=eg[ii] end;

 aff[1]:=0;aff[2]:=0;aff[3]:=0;
 for i := 0 to 6 do begin
   temp:=eef[i];    aff[2]:=aff[2] + temp; temp:=temp*i;
                    aff[1]:=aff[1] + temp; temp:=temp*i;
                    aff[3]:=aff[3] + temp                end;

 agg[1]:=0;agg[2]:=0;agg[3]:=0;
 for i := 0 to 6 do begin
   temp:=eeg[i];    agg[2]:=agg[2] + temp; temp:=temp*i;
                    agg[1]:=agg[1] + temp; temp:=temp*i;
                    agg[3]:=agg[3] + temp                end;
```

```
 LuBkSb(2,indx,aff,aa);
 LuBkSb(2,indx,agg,aa);

 temp := -4.8; {Prediction interval to 30 milliseconds}
 john := aff[2] + temp*aff[1];
 jane := agg[2] + temp*agg[1] ;

 inc(efp); if efp >42 then efp :=0;
{ writeln(jane:12:6,john:12:6,efp:12);}
;
   FirP := Fir;
   If ( (P7In and 32=0) and (Fir=False) ) then
   begin  err:=sqrt(sqr(john)+sqr(Jane));  if err<0.0002 then Fir:=True  end;
   if ((Fir = True) and (FirP = False)) then  SetPort6_1(Port6);
   if  (Fir = True) then inc(LLL);
   if  (LLL =  40) then ResetPort6_1(Port6);
   if  (LLL =  700) then begin Fir:=False; ResetPort6_1(Port6);LLL:= 0; end;
;
   if fir then john := john+ 0.002;
;
   if fir then jane := jane + 0.002;


   if (o > ANG) then xc:=$7ff else if (o < -0.05/3) then xc:=$800 else
   xc := trunc (o*40960*3) ;  DAC00(xc);

   if (p > ANG) then xc:=$7ff else if (p < -0.05/3) then xc:=$800 else
   xc := trunc (p*40960*3);  DAC01(xc);

   if (john > ANG) then xc:=$7ff else if (john < -0.05/3) then xc:=$800 else
   xc := - trunc(john*40960*3);  DAC02(xc);

   if (jane > ANG) then xc:=$7ff else if (jane < -0.05/3) then xc:=$800 else
   xc :=  trunc(jane*40960*3);  DAC03(xc);
;

   GetPort7(P7In);
   P7InP := P7In;

   {0P}  Stepper_Driver(1, AP,TPP,YP,SP);


     setPort6_0(Port6);

video[lp] := D_El1;
```

```
video_H[lp] := D_Az1;
if printit then writeln;
 inc(lp); {Increment the Loop Counter}
 if lp = 19  then
   begin
      lp:=0;
      SetPort6_0(Port6);
      GetR6(R6);  while (R6 and Bit3) = 0 do  GetR6(R6); {Synchronization}


      Bright(vc,hc,vv,hh,vvc,hhc,btns); {vcp and hcp define the spot's location}
  {    VVC:=v_iNITIAL; HHC:=h_iNITIAL;}
      spot := (19*vvc) div 511;

      vc:=vvc; hc:=hhc;
      v_d:= V_Initial - vvc;   h_d := H_Initial - hhc;
{       gotoxy(1,1); writeln(vvc:10,hhc:10,v_d:10,h_d:10);   }

      v_d:=trunc((v_d)*1.1);
      if btns<255 then begin vc:=V_Initial; hc:=H_Initial end;
      v0:=v0p;  h0:=h0p;
      v0p := -trunc(video[spot]*23800);
      h0p:=  -trunc(video_H[spot]*18800);
      diff3:=diff2;diff2:=diff1;diff1:=diff0;diff0:=v0 - h_d;
      diffh3:=diffh2 ; diffh2:=diffh1; diffh1:=diffh0; diffh0:=h0 - h_d;

      vcp:=(5*diff0+2*diff1-diff2) div 6 + 256;
      hcp := (5*diffh0+2*diffh1-diffh2) div 6 + 256;
      vcp:=-trunc(D_El1*23800) + 256;
      hcp:=-trunc(D_Az1*18800) + 256;
      P6:=$29; PutP6(P6);
      Draw(vvvv,hhhh,vcp,hcp);
      Put_Digit(range,cntr);
      P6:=$28; PutP6(P6);
if printit then begin  gotoxy(1,1) ;writeln; end;
 end;
      ReSetPort6_0(Port6);
      DAC_UPDATE;



  end;
 end. {main}
```

NO. OF COPIES	ORGANIZATION
2	DEFENSE TECHNICAL INFORMATION CENTER DTIC DDA 8725 JOHN J KINGMAN RD STE 0944 FT BELVOIR VA 22060-6218
1	HQDA DAMO FDQ D SCHMIDT 400 ARMY PENTAGON WASHINGTON DC 20310-0460
1	OSD OUSD(A&T)/ODDDR&E(R) R J TREW THE PENTAGON WASHINGTON DC 20301-7100
1	DPTY CG FOR RDA US ARMY MATERIEL CMD AMCRDA 5001 EISENHOWER AVE ALEXANDRIA VA 22333-0001
1	INST FOR ADVNCD TCHNLGY THE UNIV OF TEXAS AT AUSTIN PO BOX 202797 AUSTIN TX 78720-2797
1	DARPA B KASPAR 3701 N FAIRFAX DR ARLINGTON VA 22203-1714
1	NAVAL SURFACE WARFARE CTR CODE B07 J PENNELLA 17320 DAHLGREN RD BLDG 1470 RM 1101 DAHLGREN VA 22448-5100
1	US MILITARY ACADEMY MATH SCI CTR OF EXCELLENCE DEPT OF MATHEMATICAL SCI MADN MATH THAYER HALL WEST POINT NY 10996-1786

NO. OF COPIES	ORGANIZATION
1	DIRECTOR US ARMY RESEARCH LAB AMSRL DD 2800 POWDER MILL RD ADELPHI MD 20783-1197
1	DIRECTOR US ARMY RESEARCH LAB AMSRL CS AS (RECORDS MGMT) 2800 POWDER MILL RD ADELPHI MD 20783-1145
3	DIRECTOR US ARMY RESEARCH LAB AMSRL CI LL 2800 POWDER MILL RD ADELPHI MD 20783-1145
	ABERDEEN PROVING GROUND
4	DIR USARL AMSRL CI LP (BLDG 305)

NO. OF COPIES ORGANIZATION

3 CDR
USASOC
DSCR
AOFI RI MSI
E BROWN SOST COORDINATOR
FORT BRAGG NC 28307

2 CDR
USASOCOM
SOST T
W WILLIAMS
BLDG 102
MACDILL AFB FL 33621-5316

1 OFFICE OF SPECIAL TECH
G SHOCK
10530 RIVERVIEW RD
FT WASHINGTON MD 20744

5 CDR
US ARMY ARDEC
AMSTA CCJ
S SMALL
PICATINNY ARSENAL NJ
07806-5000

2 CDR
US ARMY ARDEC
AMSTA DSA SA
J UNTERKOFLER
M DOWNES
PICATINNY ARSENAL NJ
07806-5000

NO. OF COPIES ORGANIZATION

ABERDEEN PROVING GROUND

5 DIR USATC
STEAC AE CA
P MCCALL
D GRIFFIN
STEAC FC M
A ROSE
G NIEWENHOUS
G BREWER

7 DIR USARL
AMSRL WM BA
W D'AMICO
T BROSSEAU
B HAUG
M KREGEL
J MCLAUGHLIN
AMSRL WM MB
R KASTE
L BURTON

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 2000	Final,

4. TITLE AND SUBTITLE
The Inertial Reticle Technology (IRT) Applied to a .50-cal. M2 Heavy-Barrel Machine Gun Firing From a High-Mobility Multipurpose Wheeled Vehicle (HMMWV)

5. FUNDING NUMBERS
1L162618AH80

6. AUTHOR(S)
Timothy L. Brosseau, Mark D. Kregel, Bailey T. Haug, and John T. McLaughlin

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: AMSRL-WM-BA
Aberdeen Proving Ground, MD 21005-5066

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-2210

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)

10.SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

Motion of the muzzle of a weapon fired from a moving vehicle occurs during firing because of many factors, such as vibrations caused by the vehicle's wheels or the terrain. This motion can have adverse effects on the capabilities of the weapon to hit a target, because the shooter is unable to accurately position the muzzle of the weapon onto the target as the projectile exits the barrel. Large, heavy vehicles, such as the Abrams tank, the Bradley Fighting Vehicle, and the costly Apache helicopter, have very expensive gun turrets that are controlled by very expensive, fully stabilized gun sights to accurately position the muzzle of the weapon onto the target. However, small and lightweight vehicles, such as a small helicopter, a fast attack vehicle, or a high-mobility multipurpose wheeled vehicle (HMMWV), cannot justify such expensive gun turrets and fully stabilized sights. Therefore, to improve the accuracy of a weapon firing from a small, lightweight vehicle, the U.S. Army Research Laboratory (ARL) has developed the Inertial Reticle Technology (IRT).

This report presents the complete details of how the IRT was applied to a .50-cal. M2 heavy-barrel machine gun firing from a HMMWV, along with an analysis of stationary and moving vehicle live fire test data. Also presented are analyses of moving vehicle live fire test data from a .50-cal. M2 heavy-barrel machine gun firing from the swivel turret of a standard HMMWV, without the IRT.

14. SUBJECT TERMS
Inertial Reticle Technology, .50-cal. M2 machine gun, HMMWV, M33 ammunition, video camera, flat display monitor, quartz rate sensors, shaft angle encoders, firing solenoid

15. NUMBER OF PAGES
62

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

INTENTIONALLY LEFT BLANK.

USER EVALUATION SHEET/CHANGE OF ADDRESS

This Laboratory undertakes a continuing effort to improve the quality of the reports it publishes. Your comments/answers to the items/questions below will aid us in our efforts.

1. ARL Report Number/Author ARL-TR-2210 (Brosseau) Date of Report April 2000

2. Date Report Received _____

3. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which the report will be used.) _____

4. Specifically, how is the report being used? (Information source, design data, procedure, source of ideas, etc.) _____

5. Has the information in this report led to any quantitative savings as far as man-hours or dollars saved, operating costs avoided, or efficiencies achieved, etc? If so, please elaborate. _____

6. General Comments. What do you think should be changed to improve future reports? (Indicate changes to organization, technical content, format, etc.) _____

CURRENT ADDRESS

_____
Organization

_____
Name E-mail Name

_____
Street or P.O. Box No.

_____
City, State, Zip Code

7. If indicating a Change of Address or Address Correction, please provide the Current or Correct address above and the Old or Incorrect address below.

OLD ADDRESS

_____
Organization

_____
Name

_____
Street or P.O. Box No.

_____
City, State, Zip Code

(Remove this sheet, fold as indicated, tape closed, and mail.)
(DO NOT STAPLE)